

**MACHINE LEARNING FOR INTELLIGENT CONTROL: APPLICATION  
OF REINFORCEMENT LEARNING TECHNIQUES TO THE  
DEVELOPMENT OF FLIGHT CONTROL SYSTEMS FOR MINIATURE  
UAV ROTORCRAFT**

---

A thesis submitted in partial fulfilment of the requirements for the Degree  
of Master of Engineering in Mechanical Engineering  
in the University of Canterbury  
by Edwin Hayes  
University of Canterbury  
2013

---



## Abstract

This thesis investigates the possibility of using reinforcement learning (RL) techniques to create a flight controller for a quadrotor Micro Aerial Vehicle (MAV).

A capable flight control system is a core requirement of any unmanned aerial vehicle. The challenging and diverse applications in which MAVs are destined to be used, mean that considerable time and effort need to be put into designing and commissioning suitable flight controllers. It is proposed that reinforcement learning, a subset of machine learning, could be used to address some of the practical difficulties.

While much research has delved into RL in unmanned aerial vehicle applications, this work has tended to ignore low level motion control, or been concerned only in off-line learning regimes. This thesis addresses an area in which accessible information is scarce: the performance of RL when used for on-policy motion control.

Trying out a candidate algorithm on a real MAV is a simple but expensive proposition. In place of such an approach, this research details the development of a suitable simulator environment, in which a prototype controller might be evaluated. Then inquiry then proposes a possible RL-based control system, utilising the Q-learning algorithm, with an adaptive RBF-network providing function approximation.

The operation of this prototypical control system is then tested in detail, to determine both the absolute level of performance which can be expected, and the effect which tuning critical parameters of the algorithm has on the functioning of the controller. Performance is compared against a conventional PID controller to maximise the usability of the results by a wide audience. Testing considers behaviour in the presence of disturbances, and run-time changes in plant dynamics.

Results show that given sufficient learning opportunity, a RL-based control system performs as well as a simple PID controller. However, unstable behaviour during learning is an issue for future analysis.

Additionally, preliminary testing is performed to evaluate the feasibility of implementing RL algorithms in an embedded computing environment, as a general requirement for a MAV flight controller. Whilst the algorithm runs successfully in an embedded context, observation reveals further development would be necessary to reduce computation time to a level where a controller was able to update sufficiently quickly for a real-time motion control application.

In summary, the study provides a critical assessment of the feasibility of using RL algorithms for motion control tasks, such as MAV flight control. Advantages which merit interest are exposed, though practical considerations suggest at this stage, that such a control system is not a realistic proposition. There is a discussion of avenues which may uncover possibilities to surmount these challenges. This investigation will prove useful for engineers interested in the opportunities which reinforcement learning techniques represent.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.1.1	UASs, MAVs & Autopilots . . . . .	1
1.1.2	Issues with Traditional Control Systems . . . . .	1
1.1.3	Possible Application of Machine Learning . . . . .	2
1.1.4	Similar Research & Objectives . . . . .	3
1.2	Outline . . . . .	3
1.2.1	Problem Domain . . . . .	3
1.2.2	Thesis Summary . . . . .	4
1.2.3	Limitations . . . . .	4
<b>2</b>	<b>Background &amp; Literature</b>	<b>5</b>
2.1	Background of Machine Learning in General . . . . .	5
2.2	Background of Reinforcement Learning . . . . .	6
2.2.1	About Reinforcement Learning . . . . .	6
2.2.2	About Q-Learning . . . . .	7
2.2.3	About Eligibility Traces . . . . .	8
2.2.4	Q-Learning Function Estimation via ARBFN . . . . .	8
2.3	Reinforcement Learning for UAV Platforms . . . . .	9
2.3.1	Remaining Questions . . . . .	11
2.4	A Note on Terminology . . . . .	11
<b>3</b>	<b>Design of Micro Aerial Vehicles</b>	<b>13</b>
3.1	About Quadrotor MAVs . . . . .	13
3.1.1	General Scheme . . . . .	13
3.1.2	Basic Dynamics . . . . .	14
3.1.3	A Typical Quadrotor UAV . . . . .	15
3.1.4	Test Stand . . . . .	15
<b>4</b>	<b>Software Design</b>	<b>19</b>
4.1	Computing Software Technologies . . . . .	19
4.1.1	Java . . . . .	19
4.1.2	Reinforcement Learning Library (Teachingbox) . . . . .	20
4.1.3	J3D Robot Simulator (Simbad) . . . . .	21
4.2	Quadrotor UAV Simulator . . . . .	22
4.2.1	Encoding of Simulator State . . . . .	22
4.2.2	Freedom During Testing . . . . .	23
4.2.3	Encoding of Controller State . . . . .	24
4.2.4	Throttle Signal Mixing . . . . .	25
4.2.5	Simulator Implementation . . . . .	26
4.3	Controllers . . . . .	29

4.3.1	Traditional PID Controller . . . . .	30
4.3.2	Q-Learning Controller . . . . .	31
4.4	Integration of Components . . . . .	35
4.4.1	Modifications Required . . . . .	35
4.4.2	Typical Call Chain . . . . .	37
<b>5</b>	<b>Performance of PID Controller</b>	<b>39</b>
5.1	Experimental Details . . . . .	39
5.1.1	Test Procedure . . . . .	39
5.1.2	Performance Metrics . . . . .	40
5.2	Results . . . . .	40
5.2.1	A First Attempt . . . . .	40
5.2.2	Other Controller Gains . . . . .	42
5.2.3	Effects of Disturbance . . . . .	45
5.2.4	Handling Changes in Dynamics . . . . .	46
<b>6</b>	<b>Performance of Q-Learning Based Controller</b>	<b>53</b>
6.1	Experimental Details . . . . .	53
6.1.1	Test Procedure . . . . .	53
6.1.2	Performance Metrics . . . . .	54
6.1.3	The Term ‘Stability’ . . . . .	54
6.2	Tuning Parameters . . . . .	55
6.2.1	Initial Baseline . . . . .	55
6.2.2	Varying Learning Rate $\alpha$ . . . . .	60
6.2.3	Varying Discount Rate $\gamma$ . . . . .	66
6.2.4	Varying Eligibility Trace Decay Rate $\lambda$ . . . . .	73
6.2.5	Varying Velocity Reward Gain $K_{\dot{\theta}}$ . . . . .	81
6.2.6	Varying Displacement ARBFN Resolution $\sigma_{\theta}$ . . . . .	92
6.2.7	Varying Velocity ARBFN Resolution $\sigma_{\dot{\theta}}$ . . . . .	100
6.2.8	Varying Greediness $\epsilon$ . . . . .	107
6.2.9	Final Baseline . . . . .	112
6.3	Effects of Disturbance . . . . .	118
6.4	Handling Changes in Dynamics . . . . .	121
6.4.1	Without Disturbances . . . . .	121
6.4.2	With Disturbances . . . . .	127
6.5	Inspection of Q-Functions and Policies . . . . .	133
6.6	Assessment of Results . . . . .	137
<b>7</b>	<b>Computational Considerations</b>	<b>139</b>
7.1	Introduction . . . . .	139
7.1.1	Importance of Computational Performance . . . . .	139
7.1.2	Practical Considerations . . . . .	140
7.2	Computing Hardware Technologies . . . . .	140
7.2.1	Desktop Computer . . . . .	140
7.2.2	Embedded ARM Computer . . . . .	141
7.3	Performance Comparison . . . . .	142
7.3.1	Performance Metrics . . . . .	142
7.3.2	Experimental Procedure . . . . .	143
7.3.3	Expectations . . . . .	143
7.3.4	Results . . . . .	144
<b>8</b>	<b>Conclusions</b>	<b>147</b>

8.1	Considerations for Parameter Selection . . . . .	147
8.2	Viability for Practical Use . . . . .	148
8.3	Further Expansion & Future Work . . . . .	149
8.3.1	Longer Simulations . . . . .	150
8.3.2	More Comprehensive Test Regime . . . . .	150
8.3.3	Testing on Test Platform . . . . .	150
8.3.4	Free Rotation . . . . .	151
8.3.5	Incorporating Translational Control . . . . .	151
8.3.6	Adding Multiple Actions & Continuous Actions . . . . .	151
8.3.7	Hybrid Approaches . . . . .	152
8.4	Closing Remarks . . . . .	153
<b>A</b>	<b>Additional Q-Function and Policy Plots</b>	<b>155</b>
	<b>Bibliography</b>	<b>165</b>





# List of Figures

3.1	Illustrative diagram of simplified model of forces on quadrotor MAV. . . . .	14
3.2	Illustrative diagram of principles for movement of a quadrotor MAV. . . . .	16
3.3	The Droidworx CX-4, a fairly typical quadrotor MAV intended for industrial applications. . . . .	16
3.4	The UAV test stand. The X/Y translational arm is denoted Details A and B. The translational Z slide is denoted Detail C, and the rotational gimbal frame is denoted Detail D. . . . .	18
4.1	Diagram showing the global/earth reference frame (denoted $E$ ) and the base/vehicle reference frame (denoted $B$ ). The transformation from frame $E$ to frame $B$ is composed of a translational component (denoted $T$ ) and a rotational component (denoted $R$ ). . . . .	24
4.2	Screen-grab of simulator user interface, showing visualisation of simulated UAV.	28
4.3	UML Diagram of Simulator Components: Grey components belong to the Simbad and Teachingbox libraries; yellow components were created during this project. .	38
5.1	Individual episode lengths for PID controller ( $K_p = 0.5$ , $K_d = 5.0$ ). . . . .	41
5.2	Compiled episode lengths for PID controller ( $K_p = 0.5$ , $K_d = 5.0$ ). . . . .	41
5.3	Sample time-domain response for PID controller ( $K_p = 0.5$ , $K_d = 5.0$ ). . . . .	43
5.4	Compiled episode lengths for PID controller with varying gains. . . . .	43
5.5	Sample time-domain response for PID controller ( $K_p = 0.5$ , $K_d = 15.0$ ). . . . .	44
5.6	Sample time-domain response for PID controller ( $K_p = 0.5$ , $K_d = 10.0$ ). . . . .	44
5.7	Compiled episode lengths for PID controller in the presence of disturbances. . .	46
5.8	Sample time-domain response for PID controller with disturbances ( $K_p = 0.5$ , $K_d = 10.0$ ). . . . .	47
5.9	Individual episode lengths for PID controller ( $K_p = 0.5$ , $K_d = 10.0$ ) with modified plant dynamics. . . . .	48
5.10	Compiled episode lengths for PID controller ( $K_p = 0.5$ , $K_d = 10.0$ ) with modified plant dynamics. . . . .	48
5.11	Sample time-domain response for PID controller ( $K_p = 0.5$ , $K_d = 10.0$ ) before modifying plant dynamics. . . . .	50
5.12	Sample time-domain response for PID controller ( $K_p = 0.5$ , $K_d = 10.0$ ) after modifying plant dynamics. . . . .	50
5.13	Individual episode lengths for PID controller ( $K_p = 0.5$ , $K_d = 10.0$ ) with modified plant dynamics and disturbances. . . . .	51
5.14	Compiled episode lengths for PID controller ( $K_p = 0.5$ , $K_d = 10.0$ ) with modified plant dynamics and disturbances. . . . .	51
5.15	Sample time-domain response for PID controller ( $K_p = 0.5$ , $K_d = 10.0$ ) with disturbances before modifying plant dynamics. . . . .	52
5.16	Sample time-domain response for PID controller ( $K_p = 0.5$ , $K_d = 10.0$ ) with disturbances after modifying plant dynamics. . . . .	52

6.1	Individual episode lengths for QL controller (initial baseline). . . . .	56
6.2	Smoothed episode lengths for QL controller (initial baseline). . . . .	57
6.3	Distribution of episode lengths in Region 1 for QL controller (initial baseline). . .	58
6.4	Distribution of episode lengths in Region 2 for QL controller (initial baseline). . .	58
6.5	Distribution of episode lengths in Region 3 for QL controller (initial baseline). . .	59
6.6	Distribution of episode lengths in Region 4 for QL controller (initial baseline). . .	59
6.7	Distribution of concatenated episode lengths for QL controller (initial baseline). .	60
6.8	Sample time-domain response for QL controller (initial baseline). . . . .	61
6.9	Smoothed episode lengths for QL controller ( $\alpha = 0.5$ ). . . . .	62
6.10	Distribution of concatenated episode lengths for QL controller ( $\alpha = 0.5$ ). . . . .	62
6.11	Sample time-domain response for QL controller ( $\alpha = 0.5$ ). . . . .	64
6.12	Smoothed episode lengths for QL controller ( $\alpha = 0.1$ ). . . . .	64
6.13	Distribution of concatenated episode lengths for QL controller ( $\alpha = 0.1$ ). . . . .	65
6.14	Sample time-domain response for QL controller ( $\alpha = 0.1$ ). . . . .	66
6.15	Smoothed episode lengths for QL controller ( $\gamma = 0.95$ ). . . . .	67
6.16	Distribution of concatenated episode lengths for QL controller ( $\gamma = 0.95$ ). . . . .	68
6.17	Sample time-domain response for QL controller ( $\gamma = 0.95$ ). . . . .	69
6.18	Smoothed episode lengths for QL controller ( $\gamma = 0.99$ ). . . . .	70
6.19	Distribution of concatenated episode lengths for QL controller ( $\gamma = 0.99$ ). . . . .	71
6.20	Sample time-domain response for QL controller ( $\gamma = 0.99$ ). . . . .	72
6.21	Smoothed episode lengths for QL controller ( $\gamma = 0.85$ ). . . . .	72
6.22	Distribution of concatenated episode lengths for QL controller ( $\gamma = 0.85$ ). . . . .	74
6.23	Sample time-domain response for QL controller ( $\gamma = 0.85$ ). . . . .	74
6.24	Smoothed episode lengths for QL controller ( $\lambda = 0.9$ ). . . . .	76
6.25	Distribution of concatenated episode lengths for QL controller ( $\lambda = 0.9$ ). . . . .	76
6.26	Sample time-domain response for QL controller ( $\lambda = 0.9$ ). . . . .	77
6.27	Smoothed episode lengths for QL controller ( $\lambda = 0.95$ ). . . . .	78
6.28	Distribution of concatenated episode lengths for QL controller ( $\lambda = 0.95$ ). . . . .	78
6.29	Sample time-domain response for QL controller ( $\lambda = 0.95$ ). . . . .	79
6.30	Smoothed episode lengths for QL controller ( $\lambda = 0.7$ ). . . . .	80
6.31	Distribution of concatenated episode lengths for QL controller ( $\lambda = 0.7$ ). . . . .	81
6.32	Sample time-domain response for QL controller ( $\lambda = 0.7$ ). . . . .	82
6.33	Smoothed episode lengths for QL controller ( $K_{\dot{\theta}} = 0.2$ ). . . . .	83
6.34	Distribution of concatenated episode lengths for QL controller ( $K_{\dot{\theta}} = 0.2$ ). . . . .	83
6.35	Sample time-domain response for QL controller ( $K_{\dot{\theta}} = 0.2$ ). . . . .	85
6.36	Smoothed episode lengths for QL controller ( $K_{\dot{\theta}} = 0.35$ ). . . . .	85
6.37	Distribution of concatenated episode lengths for QL controller ( $K_{\dot{\theta}} = 0.35$ ). . . . .	86
6.38	Sample time-domain response for QL controller ( $K_{\dot{\theta}} = 0.35$ ). . . . .	88
6.39	Smoothed episode lengths for QL controller ( $K_{\dot{\theta}} = 0.5$ ). . . . .	88
6.40	Distribution of concatenated episode lengths for QL controller ( $K_{\dot{\theta}} = 0.5$ ). . . . .	89
6.41	Sample time-domain response for QL controller ( $K_{\dot{\theta}} = 0.5$ ). . . . .	90
6.42	Smoothed episode lengths for QL controller ( $K_{\dot{\theta}} = 0.1$ ). . . . .	90
6.43	Distribution of concatenated episode lengths for QL controller ( $K_{\dot{\theta}} = 0.1$ ). . . . .	91
6.44	Sample time-domain response for QL controller ( $K_{\dot{\theta}} = 0.1$ ). . . . .	93
6.45	Smoothed episode lengths for QL controller ( $K_{\dot{\theta}} = 0.0$ ). . . . .	93
6.46	Distribution of concatenated episode lengths for QL controller ( $K_{\dot{\theta}} = 0.0$ ). . . . .	94
6.47	Sample time-domain response for QL controller ( $K_{\dot{\theta}} = 0.0$ ). . . . .	95
6.48	Smoothed episode lengths for QL controller ( $\sigma_{\theta} = 256$ ). . . . .	96
6.49	Distribution of concatenated episode lengths for QL controller ( $\sigma_{\theta} = 256$ ). . . . .	96
6.50	Sample time-domain response for QL controller ( $\sigma_{\theta} = 256$ ). . . . .	98
6.51	Smoothed episode lengths for QL controller ( $\sigma_{\theta} = 80$ ). . . . .	98

6.52	Distribution of concatenated episode lengths for QL controller ( $\sigma_\theta = 80$ ).	99
6.53	Sample time-domain response for QL controller ( $\sigma_\theta = 80$ ).	101
6.54	Smoothed episode lengths for QL controller ( $\sigma_\theta = 32$ ).	101
6.55	Smoothed episode lengths for QL controller ( $\sigma_{\dot{\theta}} = 128$ ).	103
6.56	Distribution of concatenated episode lengths for QL controller ( $\sigma_{\dot{\theta}} = 128$ ).	103
6.57	Sample time-domain response for QL controller ( $\sigma_{\dot{\theta}} = 128$ ).	104
6.58	Smoothed episode lengths for QL controller ( $\sigma_{\dot{\theta}} = 32$ ).	105
6.59	Distribution of concatenated episode lengths for QL controller ( $\sigma_{\dot{\theta}} = 32$ ).	105
6.60	Sample time-domain response for QL controller ( $\sigma_{\dot{\theta}} = 32$ ).	106
6.61	Smoothed episode lengths for QL controller ( $\epsilon = 0.2$ ).	108
6.62	Distribution of concatenated episode lengths for QL controller ( $\epsilon = 0.2$ ).	108
6.63	Sample time-domain response for QL controller ( $\epsilon = 0.2$ ).	110
6.64	Smoothed episode lengths for QL controller ( $\epsilon = 0.01$ ).	110
6.65	Distribution of concatenated episode lengths for QL controller ( $\epsilon = 0.01$ ).	111
6.66	Sample time-domain response for QL controller ( $\epsilon = 0.01$ ).	113
6.67	Individual episode lengths for QL controller (final baseline).	113
6.68	Smoothed episode lengths for QL controller (final baseline).	114
6.69	Distribution of episode lengths in Region 1 for QL controller (final baseline).	114
6.70	Distribution of episode lengths in Region 2 for QL controller (final baseline).	115
6.71	Distribution of episode lengths in Region 3 for QL controller (final baseline).	115
6.72	Distribution of episode lengths in Region 4 for QL controller (final baseline).	116
6.73	Distribution of concatenated episode lengths for QL controller (final baseline).	116
6.74	Sample time-domain response for QL controller (final baseline).	118
6.75	Individual episode lengths for QL controller with disturbances.	119
6.76	Smoothed episode lengths for QL controller with disturbances.	119
6.77	Distribution of concatenated episode lengths for QL controller with disturbances.	120
6.78	Sample time-domain response for QL controller with disturbances.	122
6.79	Individual episode lengths for QL controller with modified plant dynamics.	122
6.80	Smoothed episode lengths for QL controller with modified plant dynamics.	124
6.81	Distribution of concatenated episode lengths for QL controller, before modifying plant dynamics.	124
6.82	Distribution of concatenated episode lengths for QL controller, after modifying plant dynamics.	125
6.83	Sample time-domain response for QL controller, before modifying plant dynamics.	126
6.84	Sample time-domain response for QL controller, after modifying plant dynamics.	126
6.85	Individual episode lengths for QL controller with disturbances and modified plant dynamics.	128
6.86	Smoothed episode lengths for QL controller with disturbances and modified plant dynamics.	128
6.87	Distribution of concatenated episode lengths for QL controller with disturbances, before modifying plant dynamics.	129
6.88	Distribution of concatenated episode lengths for QL controller with disturbances, after modifying plant dynamics.	130
6.89	Sample time-domain response for QL controller with disturbances, before modifying plant dynamics.	131
6.90	Sample time-domain response for QL controller with disturbances, after modifying plant dynamics.	132
6.91	A typical Q-Function for the X (roll) axis, after 5 million steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.	134

6.92	A typical policy function for the X (roll) axis, after 5 million steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	134
6.93	Another typical Q-Function for the X (roll) axis, after 5 million steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	135
6.94	Another typical policy function for the X (roll) axis, after 5 million steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	135
6.95	A typical Q-Function for the X (roll) axis, after 5 million steps (using $\alpha = 0.3$ , $\gamma = 0.8$ and $\lambda = 0.7$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	136
6.96	A typical policy function for the X (roll) axis, after 5 million steps (using $\alpha = 0.3$ , $\gamma = 0.8$ and $\lambda = 0.7$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	136
7.1	Time between Simulation Iterations . . . . .	145
A.1	A typical Q-Function for the X (roll) axis, after 10,000 steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). X-axis in Radians, Y-axis in Radians per time-step. . . . .	156
A.2	A typical policy function for the X (roll) axis, after 10,000 steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	156
A.3	A typical Q-Function for the X (roll) axis, after 20,000 steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	157
A.4	A typical policy function for the X (roll) axis, after 20,000 steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	157
A.5	A typical Q-Function for the X (roll) axis, after 30,000 steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	158
A.6	A typical policy function for the X (roll) axis, after 30,000 steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	158
A.7	A typical Q-Function for the X (roll) axis, after 50,000 steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	159
A.8	A typical policy function for the X (roll) axis, after 50,000 steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	159
A.9	A typical Q-Function for the X (roll) axis, after 100,000 steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	160
A.10	A typical policy function for the X (roll) axis, after 100,000 steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	160
A.11	A typical Q-Function for the X (roll) axis, after 200,000 steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	161

A.12	A typical policy function for the X (roll) axis, after 200,000 steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	161
A.13	A typical Q-Function for the X (roll) axis, after 300,000 steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	162
A.14	A typical policy function for the X (roll) axis, after 300,000 steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	162
A.15	A typical Q-Function for the X (roll) axis, after 500,000 steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	163
A.16	A typical policy function for the X (roll) axis, after 500,000 steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	163
A.17	A typical Q-Function for the X (roll) axis, after 1000,000 steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	164
A.18	A typical policy function for the X (roll) axis, after 1000,000 steps (using $\alpha = 0.2$ , $\gamma = 0.9$ and $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step. . . . .	164



# List of Tables

5.1	Episode length statistics for experiments varying PID controller gains (aggregate results for each experiment are shown shaded).	42
5.2	Episode length statistics for experiments with and without disturbances (aggregate results for each experiment are shown shaded).	45
5.3	Episode length statistics for experiments modifying plant dynamics (aggregate results).	47
6.1	Initial set of tunable parameters for QL controller.	55
6.2	KPIs for QL controller (initial baseline).	57
6.3	Tunable parameters for QL controller ( $\alpha = 0.5$ ).	61
6.4	KPIs for QL controller ( $\alpha = 0.5$ ).	63
6.5	Tunable parameters for QL controller ( $\alpha = 0.1$ ).	63
6.6	KPIs for QL controller ( $\alpha = 0.1$ ).	65
6.7	Tunable parameters for QL controller ( $\gamma = 0.95$ ).	67
6.8	KPIs for QL controller ( $\gamma = 0.95$ ).	67
6.9	Tunable parameters for QL controller ( $\gamma = 0.99$ ).	69
6.10	KPIs for QL controller ( $\gamma = 0.99$ ).	70
6.11	Tunable parameters for QL controller ( $\gamma = 0.85$ ).	71
6.12	KPIs for QL controller ( $\gamma = 0.85$ ).	73
6.13	Tunable parameters for QL controller ( $\lambda = 0.9$ ).	75
6.14	KPIs for QL controller ( $\lambda = 0.9$ ).	77
6.15	Tunable parameters for QL controller ( $\lambda = 0.9$ ).	77
6.16	KPIs for QL controller ( $\lambda = 0.95$ ).	79
6.17	Tunable parameters for QL controller ( $\lambda = 0.7$ ).	79
6.18	KPIs for QL controller ( $\lambda = 0.7$ ).	80
6.19	Tunable parameters for QL controller ( $K_{\dot{\theta}} = 0.2$ ).	82
6.20	KPIs for QL controller ( $K_{\dot{\theta}} = 0.2$ ).	84
6.21	Tunable parameters for QL controller ( $K_{\dot{\theta}} = 0.35$ ).	84
6.22	KPIs for QL controller ( $K_{\dot{\theta}} = 0.35$ ).	86
6.23	Tunable parameters for QL controller ( $K_{\dot{\theta}} = 0.5$ ).	87
6.24	KPIs for QL controller ( $K_{\dot{\theta}} = 0.5$ ).	89
6.25	Tunable parameters for QL controller ( $K_{\dot{\theta}} = 0.1$ ).	89
6.26	KPIs for QL controller ( $K_{\dot{\theta}} = 0.1$ ).	91
6.27	Tunable parameters for QL controller ( $K_{\dot{\theta}} = 0.0$ ).	92
6.28	KPIs for QL controller ( $K_{\dot{\theta}} = 0.0$ ).	94
6.29	Tunable parameters for QL controller ( $\sigma_{\theta} = 256$ ).	95
6.30	KPIs for QL controller ( $\sigma_{\theta} = 256$ ).	97
6.31	Tunable parameters for QL controller ( $\sigma_{\theta} = 80$ ).	97
6.32	KPIs for QL controller ( $\sigma_{\theta} = 80$ ).	99
6.33	Tunable parameters for QL controller ( $\sigma_{\theta} = 32$ ).	100

6.34	Tunable parameters for QL controller ( $\sigma_{\dot{\theta}} = 128$ ). . . . .	102
6.35	KPIs for QL controller ( $\sigma_{\dot{\theta}} = 128$ ). . . . .	104
6.36	Tunable parameters for QL controller ( $\sigma_{\dot{\theta}} = 32$ ). . . . .	104
6.37	KPIs for QL controller ( $\sigma_{\dot{\theta}} = 32$ ). . . . .	106
6.38	Tunable parameters for QL controller ( $\epsilon = 0.2$ ). . . . .	107
6.39	KPIs for QL controller ( $\epsilon = 0.2$ ). . . . .	109
6.40	Tunable parameters for QL controller ( $\epsilon = 0.01$ ). . . . .	109
6.41	KPIs for QL controller ( $\epsilon = 0.01$ ). . . . .	111
6.42	Final set of tunable parameters for QL controller. . . . .	112
6.43	KPIs for QL controller (final baseline). . . . .	117
6.44	KPIs for QL controller with disturbances. . . . .	120
6.45	KPIs for QL controller, before modifying plant dynamics. . . . .	123
6.46	KPIs for QL controller, after modifying plant dynamics. . . . .	125
6.47	KPIs for QL controller with disturbances, before modifying plant dynamics. . . .	127
6.48	KPIs for QL controller with disturbances, after modifying plant dynamics. . . . .	129
7.1	Summary of Vital Statistics for Computing Hardware Technologies. . . . .	141
7.2	Total execution time for one million steps of simulation. . . . .	144



# List of Abbreviations

ARBFN Adaptive Radial-Basis Function Network

CPU Central Processing Unit

DDR Double Data Rate

EWMA Exponentially Weighted Moving Average

FVI Fitted Value Iteration

GPU Graphics Processing Unit

ISM Integral Sliding Mode

KPI Key Performance Indicator

MAV Micro Aerial Vehicle

ML Machine Learning

PID Proportional Integral Derivative

QL Q-Learning

RBF Radial Basis Function

RL Reinforcement Learning

SATA Serial AT (Advanced Technology) Attachment

SD-RAM Synchronous Dynamic Random-Access Memory

UAS Unmanned Aerial System

UAV Unmanned Aerial Vehicle

UML Unified Modelling Language



# Chapter 1

## Introduction

In this chapter, some limitations with the conventional methodology for the design of control systems, particularly relevant in the design of flight control systems of Micro Aerial Vehicles, are considered. The concept of using Reinforcement Learning techniques to address these issues is introduced. The motivation, nature, scope and limitations of this project are then discussed.

### 1.1 Motivation

#### 1.1.1 UASs, MAVs & Autopilots

The field of Unmanned Aerial Systems (UAS') is receiving considerable attention, both in terms of academic research <sup>1</sup>, and possible applications [17]. Of particular interest are smaller systems, commonly referred to as Micro Aerial Vehicles (MAVs). Because of their reduced size and lower cost, they are suitable for a wide range of tasks for which larger vehicles are unsuited [2] [15] [41]. However, many difficulties remain to be addressed.

Any UAS requires some form of flight control system (commonly referred to as an 'autopilot'), which operates the vehicle in lieu of a human pilot. Whilst many examples of successful autopilots exist [14] [9] [13], aspects of these require further refinement. Many of these aspects are of particular importance when considered with respect to MAVs.

For example: the low cost of MAVs ensures that they tend to be deployed in large numbers; and the small size of MAVs means they tend to be used for a particular specialized task. This results in the development cost of a MAV representing a large proportion of its total cost. Accordingly, it is desirable to give consideration to possible methods for reducing the complexity (and hence cost) of developing a flight control system for a new MAV.

#### 1.1.2 Issues with Traditional Control Systems

Development of a flight control system for a MAV can initially seem simple. However, as performance requirements increase, a basic model of the dynamics of the vehicle may prove

---

<sup>1</sup>Google Scholar finds over two thousand relevant articles in the part year alone.

insufficient, requiring more complicated effects be considered. The difficulty of designing such a control system can rapidly escalate.

Additionally, a flight control system developed with conventional methods relies on the developer having available a complete model of the dynamics of the vehicle (they are ‘model-based’). If such a model is unavailable, the dynamics must be characterized through testing before work can begin on the control system. This extends the development time associated with the control system.

Further, if some modification to the vehicle is made later, then the validity of the original dynamics model may be reduced. Testing and design process must be repeated to compensate for the change. If some change to the dynamics of the vehicle occurs inadvertently during the course of operation, then the control system may become unable to control the vehicle. An example being additional payload shifting the centre of mass of the vehicle sufficiently that the controller became unstable.

It can be seen that most of the concerns stem from requiring an accurate model of the plant before a control system can be implemented. This is no mere inconvenience, as from the perspective of the user, there is no *desire* for a model of the dynamics; what is wanted is a working control system. This situation requires those characteristics of the vehicle which will impact the dynamics must be finalized early in the development process.

If a control system were available which did not require an accurate model of the MAV, but was able to determine how to control the vehicle with similar performance as a traditional (model-based) controller, this specific control system could have a significant advantage in terms of development time, cost and convenience. These benefits would be further cemented if the controller was able to compensate for any changes to the dynamics of the vehicle during operation.

### 1.1.3 Possible Application of Machine Learning

Although a controller that requires absolutely no knowledge of the dynamics of the plant whilst displaying perfect stability sounds unattainable, machine learning algorithms do seem to offer some characteristics which may address some of the questions outlined above.

Specifically, reinforcement learning (RL) algorithms provide a number of characteristics which could partly address the need for an exact model and handling dynamics changes: RL can control an agent when a model of the environment is unavailable; RL can control an agent without an existing ‘expert system’ to learn from; and RL can compensate for changes in the environment as they are encountered. This suggests that a control system using RL principles could display characteristics which would substantially reduce development effort: an RL could plausibly be able to ‘learn’ the dynamics of the vehicle, alleviating the need to develop a complex model as an intermediate step; and may be able to compensate for changes in dynamics as they occur, with less intervention by the developer.

One of several immediate questions is: how does the ‘learning’ occur? In many control systems, stability is critically important. Permitting blind experimentation which might result in a crash is unacceptable. Notwithstanding possible restrictions are not sufficient reason for the concept not to be given some consideration.

### 1.1.4 Similar Research & Objectives

Whilst extensive research has been performed into using reinforcement learning algorithms for control tasks, this exploration tends to be overly academic in its scope (examples specific to UAV applications are considered in §2.3).

Investigations have considered the theoretical performance of the algorithms involved, or demonstrated performance on a highly contrived control task. This kind of information is of limited benefit to an engineer uninterested in developing new RL algorithms, but interested in addressing a specific practical control problem.

Rather, this project approaches the use of reinforcement learning techniques in control system design from that of an embedded systems engineer required to implement a control system; in this case for a flight control task.

Accordingly, this project provides a detailed account of setting up a simple reinforcement learning control system, and offers analysis of the practical consequences of the performance of that system. It is envisaged that this will be a useful reference point at which other engineers can begin, because it provides:

- A quick assessment of the validity of an RL control system for motion control tasks. Other works make it difficult for an engineer without any experience in RL systems to appreciate the levels of performance current algorithms offer, and thus to know whether such control systems are suitable for a given task.
- An easily accessible guide to the basic steps involved in developing a RL based control system, regardless of the specific algorithms used or nature of the control plant. A valuable starting point for engineers, whether or not they use the same tools, design paradigms or devices.
- Basic qualitative grounds for comparison with their own attempts to utilize machine learning. The nature of RL algorithms makes it difficult for those not intimately familiar with using RL to know how to interpret their results (or even if their software is working properly!)

## 1.2 Outline

### 1.2.1 Problem Domain

This project concentrates on the development of a reinforcement-learning based flight control system for a Micro Aerial Vehicle (MAV): specifically, a quadrotor. Quadrotors have become ubiquitous in recent years; literature on high performance flight control systems for quadrotors is readily available (for example, by Hoffman [34]), making comparisons easy.

The enterprise seeks to develop a prototype control system which uses reinforcement-learning algorithms to stabilize the vehicle in flight, and compare the performance of this control system against the performance of a more conventional control system. *Practical* suitability being the focus.

### 1.2.2 Thesis Summary

This thesis is composed of eight chapters. This first chapter serves as an introduction to the project. The second chapter offers a brief overview of machine learning and reinforcement learning theory, and reviews relevant existing literature. The third chapter provides more information regarding Micro Aerial Vehicles in general and as they pertain to this project. The fourth chapter describes the development of the software associated with this project, including the control algorithms used; this would be relevant to a reader who wishes to either recreate some of the software developed for this project, or understand how the software works so as to allow better comparison with their own results. The fifth chapter details experiments which were performed for comparative purposes, to characterise the performance of a conventional PID controller. The sixth chapter features experiments which were performed to characterise the performance of the prototype reinforcement-learning controller and reports the substantive results from the project. The seventh chapter addresses the concern regarding the viability of possible control algorithms: their ability to be implemented given limited computational resources. Finally, the eighth chapter concludes, summarizing of the results of the project, how this translates into the viability of using reinforcement-learning as a tool for motion control systems, and proposes future work to continue this line of research.

### 1.2.3 Limitations

The nature of a UAS means that the consequences of failure of the flight control system are significant; a controller failure will typically result in either loss of, or irreparable damage to, the vehicle. Whilst the cost of MAVs is low compared with larger vehicles, it remains inadvisable to test a prototype control system on a real vehicle until there is confidence the control system will work as intended. With this in mind, this project will make use of a computer simulation of a MAV, rather than physical plant. If the results of simulated tests are promising, additional testing using a real vehicle may be considered for the future.

A free-flying MAV has six degrees of freedom (three translational axes, plus three rotational). It is likely there is a high degree of coupling between the dynamics of each: something which would need to be taken into account by a control system in order to guarantee high performance. However, literature on reinforcement-learning makes reference to the “curse of dimensionality”: as the number of dimensions in the environment increases, the time taken to search for a suitable policy increases exponentially. To *fully* demonstrate the viability of a reinforcement-learning control system would require controlling all six degrees-of-freedom at once, to encapsulate coupled behaviour, but the increased learning time is impractical given the limited computing resources and time available to this project.

Accordingly, this investigation will concentrate on a simplified control problem: the translational motion of the simulated quadrotor will be restricted, leaving only the three rotational axes. Further, each axis will be considered independently (no coupled dynamics). This makes the control task significantly simpler; control of such a system using a common PID controller would be straightforward for any experienced controls engineer. This approach is beneficial, since it makes comparative evaluation of the performance of a given algorithm easier.

These limitations are carefully considered, to maximise the outcome of the project in addressing the objectives set out in §1.1.4.

## Chapter 2

# Background & Literature

In this chapter, a summary of the key facets of Machine Learning and Reinforcement Learning is offered, particularly with respect to those algorithms which will be involved in this project. An overview of existing literature relevant to this domain is then discussed. The limitations of existing works from the perspective of a control systems engineer are outlined.

### 2.1 Background of Machine Learning in General

Machine learning (ML) is a broad field, encompassing a wide range of theory and algorithms, and the use of these tools to address an equally wide range of applications. Speech recognition, classification of scientific observations, and learning to play board games, are but a few examples. A general definition which can be used to determine whether a computer programme utilises ‘machine learning’ is given by Mitchell [42]:

A computer program is said to **learn** from experience  $E$  with respect to some class of tasks  $T$  and performance measure  $P$ , if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ .

Given some consideration, it is clear that this definition could include a broad variety of tasks, given suitably broad characterisations of  $E$ ,  $T$  and  $P$ . In fact, the question of whether or not machine learning is an appropriate solution to a particular programming task, and then the design of the solution itself, largely condenses around the specification of these three parameters.

Following the selection of the parameters  $E$ ,  $T$  and  $P$  which characterises the overall operation of the learning process, a ‘target function’ may be defined, which essentially embodies the task being learned. An appropriate method through which to represent this function is chosen, followed by an algorithm which will be used to approximate the function: it is this algorithm which could be considered to be ‘performing’ the act of learning.

The benefits of machine learning are self evident: the developer does not need to programme the complete range of desired functionality of a piece of software. This is extremely useful in cases where this desired functionality is not fully known, or is unable to be explained in a form which can be readily converted into a computer programme. It is also helpful where the functionality of a piece of software will be required to change over time. However, machine learning also presents a number of issues which need consideration: the most commonly encountered is the

‘curse of dimensionality’ (a term first coined by Bellman [24]); increasing the number of variables in the ‘target function’ may result in an algorithm being required to search through a ‘hypothesis space’ which grows exponentially; this can mean the computational load of a proposed algorithm is too great to be implemented practically. In some cases, assumptions may be made, to reduce the size of the solution space being searched, to a manageable level. However, such issues mean that machine learning is not suitable for every application.

As mentioned, machine learning is a broad field; the range of machine learning applications and algorithms can be taxonomized using a number of possible categories. Possibilities are suggested by Sewell [46], and Alpaydin [19] provides a number of examples.

## 2.2 Background of Reinforcement Learning

### 2.2.1 About Reinforcement Learning

The particular form of Machine Learning of interest in this project is ‘Reinforcement Learning’. Reinforcement learning differs from ‘supervised learning’ algorithms, in that the software is not provided with examples from which to draw. Instead, the algorithm learns which actions to take based on actions it has performed in the past, and how good the outcomes of those actions were perceived to be.

Both Mitchell [42] and Alpaydin [19] provide chapters discussing Reinforcement Learning. However, the definitive introduction is considered to be the textbook by Sutton & Barto [50].

Even as a subset of machine learning, there are a very wide range of reinforcement learning algorithms. However, all reinforcement learning solutions share a number of common elements: the complete system can be considered to be comprised of an *agent*, and the *environment* in which the agent operates. The agent perceives the current *state* of the environment, and then selects an *action* to perform. This action alters the state of the environment. The objective of the agent is to learn which series of actions brings about some favourable outcome in the environment. To facilitate this, the agent uses a reward function, which maps the observed state of the environment (or state/action pair) to a single scalar (the *reward*) denoting the desirability of being ‘in’ this state. The agent uses the reward values it observes to select an appropriate action, through the use of a *value function*; the agent seeks to maximize the total value of the rewards it receives. The value function is used to estimate the total future reward which the agent can expect to achieve from a given starting state. The agent estimates the values of states (or state/action pairs) and uses these to develop a *policy*: a function which maps from observed states to actions the agent should take, in order to maximise total rewards received. Some reinforcement learning agents may also utilise an internal model of the environment which they use to determine their policy.

Initially, an agent has no knowledge of the environment in which it is operating. It must hence first ‘explore’ the environment in order to discover which states are most valuable. Most commonly, this is achieved by selecting an action at random. Once the agent has learned that some states are more valuable than others, it may instead perform ‘exploitive’ behaviour; follow a policy which favours those states which are known to be most valuable. Whilst exhibiting ‘exploitive’ behaviour, the agent will not visit any states other than those which are necessary as part of the agent’s current policy. This factor needs to be considered. A balance between exploratory and exploitive behaviour must be determined, because an agent which follows a



strictly deterministic policy whilst failing to fully explore the complete state space of the environment may become trapped in local extrema. The manner in which the agent chooses which action to perform, and hence selects between exploratory and exploitive behaviour is generally called the ‘action selection’ method.

### 2.2.2 About Q-Learning

Of the wide range of reinforcement learning algorithms which are available, one of the most common is Q-Learning. It is the intention of this project to avoid the internal workings of the algorithms being used, in favour of a black-box treatment; this being more representative of the manner in which a traditional controls engineer might use reinforcement learning components. Nevertheless, a brief outline of how the Q-learning algorithm proceeds is presented.

The heart of the Q-Learning scheme is the creation of a *Q-Function* (or state/action-function, if one prefers to avoid the Q- prefix, which can be confusing). This records the *value* (as specified by the algorithm’s value function) of the agent taking a specific action whilst in a specific state; hence, the Q-Function maps state/action pairs to scalar values:

$$Q : (s, a) \rightarrow \mathbb{R} \quad (2.1)$$

The value function estimates the value of a state/action pair as being the immediate reward of the state/action pair, plus the future reward from the *next* state/action pair. This will be chosen after the current action is performed and the agent finds itself in a new state, reduced by some factor  $\gamma$  to account for immediate rewards being more preferable to an equivalent reward some time in the future. Hence, at each iteration  $t$ , the Q-Function is updated:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot (r_{t+1} + \gamma \cdot Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \quad (2.2)$$

Where  $s_t$  denotes the state at iteration  $t$ ,  $a_t$  denotes the action taken in iteration  $t$ ,  $\alpha$  denotes the ‘learning rate’ (which determines how aggressively the existing estimate of the Q-value is modified at each update),  $r_t$  denotes the immediate reward attained in iteration  $t$ , and  $\gamma$  denotes the discount factor (as mentioned above). This requires knowledge of which  $a_{t+1}$  (the action the agent will take *next* iteration), which is unavailable (though the state  $s_{t+1}$  *is* available, because one can wait to update the Q-function at each iteration until just *after* the action  $a_t$  has been performed). However, since the agent is trying to optimize the reward value it will receive, the Q-function is updated to assume the agent will generally take the best of the available actions:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot (r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \quad (2.3)$$

Where  $\max_a Q(s, a)$  denotes the maximum value of the Q-value for each of the actions  $a$  which might be taken in state  $s$ . Initially, the Q-function is initialised to some arbitrary values (in this case, as a method of maximising exploration of the state space, the values are initialised to a value higher than will be attainable normally, ensuring the agent attempts to visit them). Then, in each iteration, the Q-function is used to determine a policy for the agent to follow: the agent finds the highest valued state/action pair for the current state, and chooses to perform the matching action. This leads the agent to a new state. The Q-function value for the state/action

pair which was last used is then updated. In this manner, the agent will populate the Q-function with values which provide an estimate of how to best maximize the received reward.

One notable thing about the Q-Learning algorithm is that it is an ‘off-policy’ learner: it does not matter if the agent follows the policy which is derived from the Q-function. The agent may choose to completely ignore the policy it has created and instead do something else. The learner will continue to update the Q-function throughout. This is useful to allow insertion of additional exploratory behaviour, or support more complicated controller designs.

### 2.2.3 About Eligibility Traces

The algorithm used in this project is a slight variation on a conventional Q-learning algorithm, through the addition of eligibility traces. Conventional Q-learning is a type of temporal-difference learning; at each iteration, the Q-function is updated based upon both the immediate reward, and the reward from one time-step in the future. In contrast, Monte Carlo learning methods would utilize a value function which is calculated based upon *all* the future rewards (with each given an diminishing weighting to favour immediate reward over future reward). The latter obviously requires either waiting forever, or until the learning process ends; this is inconvenient. This operation of calculating the value of a series of past rewards is known as ‘backing up’. Hence, Q-learning uses 1-step backups, whilst a pure Monte Carlo method algorithm would use  $\infty$ -step backups.

Eligibility traces provide a mechanism for choosing an intermediate degree of backing up. The mechanism by which eligibility traces operate is somewhat obscure, and will not be detailed here (see §7 of Sutton’s text [50]). Essentially, each state/action pair is allocated a counter which is incremented when that pair is ‘visited’ by the agent, and which decays proportionally to a parameter  $\lambda$  at each iteration. This allows the Q-value for each state/action pair to be updated according to how far along the ‘backup chain’ it is found.

### 2.2.4 Q-Learning Function Estimation via ARBFN

For small state/action spaces, it is convenient for the Q-function to be represented simply as a table of values. This arrangement is fast and convenient, but requires sufficient storage space. For larger state/action spaces, tabular representation becomes impractical. This is of particular concern for this project, because one is interested in implementing a controller that will run in an embedded environment, offering limited memory.

An alternative to tabular representation is to use a function approximator. In this project, an Adaptive Radial-Basis Function Network (ARBFN) is used as a function approximator in each agent.

A radial basis function (RBF) yields a value which depends solely upon distance from some origin or centre [15]. Linear combination of such functions through a neural-network type linear combination approach yields an ‘RBF network’ which can be used as a function approximator [44]. In this case, each individual radial basis function has Gaussian shape. The network is normalised, and becomes ‘adaptive’ through a rule which adds an additional term (or ‘node’) to the network when the distance between the current position and all the existing RBF centres is greater than a predetermined threshold value. This threshold may be used to determine the ‘resolution’ with which the network approximates the parent function. In operation here, the network parameters are trained to approximate the Q-function through gradient descent.

One should note that under some circumstances, reinforcement learning algorithms may display instability when used in conjunction with function approximators. Although the particular configuration used in this project has not been identified as being at risk of instability, this is an area of ongoing research [49].

## 2.3 Reinforcement Learning for UAV Platforms

The concept of using reinforcement learning techniques for the control of unmanned aerial vehicles is not new; it has received sufficient attention that references to the approach can be found outside academia. A post by Anderson [20] on popular hobbyist website ‘DIYDrones’ is representative of the attention machine learning algorithms for motion control are receiving. Conveniently, this post is also demonstrative of one of the issues this project aims to redress: from a brief inspection of this (non-technical) article, it is difficult to determine what level of performance is actually attainable from current machine learning control systems.

Much of the research reported externally regarding machine learning in a UAV context is associated with high-level trajectory control or planning. In such examples, low level stability of the aircraft is implemented through the use of a more traditional controller design, with the machine learning aspects being occupied by learning how to complete a specific trajectory (such as acrobatics) or for learning how to navigate to a specific location (such as collision avoidance tasks). These types of controllers are generally well suited to machine learning techniques, because any initial instability in the controller results in poor performance, rather than risking catastrophic loss of the aircraft.

Schoellig’s work [45] developed an optimization based iterative learning control system for open-loop swing-up of an inverted pendulum. This technique has since been used for trajectory control of quadrotor UAVs. In the same research group at ETH Zurich, Lupashin [39] developed a simple learning strategy which is able to guide a quadrotor through a series of flips. The system is able to demonstrate online learning of acrobatics, using careful selection of initial conditions, and because outside of each discrete acrobatic event, control is able to be handed back to an existing low level PID control system. Thus the vehicle is stabilized so that each iteration begins from a known stationary configuration, regardless of the performance of the learning controller.

Bower and Naiman [26] demonstrated a simulated experiment where a fixed wing glider would navigate to stay aloft by making use of thermals detected through changes in vertical velocity of the aircraft. This work clearly demonstrates that in the case of an aircraft which is stabilised at a low level, it is able to display autonomous behaviour learned online through exploration of a space according to simple policy.

Valasek’s 2008 work [52] (building upon previous papers) develops a reinforcement learning based system for mission adaptation through morphing; the dynamics of the vehicle are modified through changes in the shape of the vehicle, so as to allow a single aircraft to be used for diverse mission profiles. In this case, a Q-learning algorithm (in conjunction with function approximation techniques such as K-nearest neighbour) was used to perform online learning during simulated flights, in order to ‘morph’ the aircraft to minimize costs such as fuel consumption. During flight, low level control of the vehicle is performed by a separate control system, which is able to adapt to the changes in dynamics introduced by the policy of the learning component.

However, of more particular relevance to this project, is existing work that has investigated the viability of reinforcement learning algorithms when applied to low-level motion control of UAV rotorcraft:

Bagnell’s 2001 experimentation [21] concentrates on a traditional (cyclic) helicopter UAV rather than the multirotors of interest to this project. However the work offers an excellent summary of the issues involved in applying machine learning techniques to motion control problems, including the issue of controller stability during training. Only pitch, roll, and X/Y translational axes are considered, with yaw and altitude held constant to reduce coupled dynamics. Bagnell uses a policy search algorithm for off-line learning based on data collected from flights where the vehicle is under manual pilot control. Experimental results show that the RL controller is able to display hover performance which is superior to that of the pilot from whom the original flight data was captured. Extension to include performing the inner loop policy search in online mode is mentioned as an area for future research.

Ng’s 2004 work [43] also concentrates on a traditional helicopter UAV. In contrast to Bagnell’s, the work extends to position control in all six axes, and is then further extended to allow trajectory control, allowing the demonstration of some basic manoeuvres. As with Bagnell’s research, learning is performed on off-line data captured from flights controlled by a human pilot. In this case, the PEGASUS learning algorithm is used. Once again, in experimental tests the algorithm demonstrates control performance which is notably superior to that of the original human pilot.

Waslander’s 2005 study [53] compares the performance of a reinforcement learning algorithm which learns a model of the vehicle dynamics (using locally weighted linear regression followed by policy iteration) against an Integral Sliding Mode based controller. Waslander concentrates on altitude control for a quadrotor UAV, rather than the attitude control problem addressed in this project; the altitude control task suffers from complex dynamics and very noisy sensor inputs. The performance of the RL control system is shown to be approximately comparable to that of the ISM controller, and able to stably control the altitude of the vehicle through step inputs despite sensor noise and non-linear dynamics. However, this learning is performed in an off-line context using existing flight data; it is not a direct replacement for a conventional control system.

Engel’s work [31] offers a fairly comprehensive overview of reinforcement learning, and its application to the flight control of a traditional helicopter. A number of RL algorithms, including Q-learning, are evaluated in an online regime on a episodic simulation of a helicopter in all six degrees of freedom. This work is similar in intent to this project, but lacks comprehensive details of the time-domain behaviour which is displayed by the controllers, and the manner in which their performance varies throughout the learning process.

In Bou-Ammar’s research [25] from 2010, a reinforcement learning algorithm using fitted value iteration (FVI), is compared to a bespoke non-linear controller design, for the purposes of stabilising a quadrotor UAV. Success is defined as the controller restoring the aircraft to a stationary position after an input disturbance. This is essentially the same task considered in this project. The FVI based RL controller is shown to deliver a stable step response (though non-linear controller’s performance remains superior). The paper makes mention of the motivation that utilisation of an RL controller would relieve the need for a designer with experience in non-linear control system design. However, whilst the final step response result of the RL controller is shown, no details of the training period required for the controller to achieve this level of performance are included in the conference paper. Considering that need for training is one of the obvious drawbacks of a RL based approach, making practical decisions regarding the viability of such a control system is difficult from this information.

### 2.3.1 Remaining Questions

Existing research uses RL algorithms which are somewhat application specific. In fact details of assumptions made in developing a structure for the model which is to be learnt tend to be the bulk of the published works. By making assumptions about the form of the environment's dynamics, or restricting which policies are considered, the search space may be shrunk considerably. This essentially trades learning time and stability for development time and portability between applications.

Further, the scope of existing works is generally restricted to the operation of RL algorithms in an off-line context.

These factors, if not considered, may lead a cursory investigation into RL techniques to offer an unrealistic expectation of how a RL algorithm might perform in an online regime, directly 'out of the box'. The lack of available literature on the performance of relatively general algorithms in control applications, and on RL performance under on-line conditions, is an issue which this project aims to address.

Beyond this, contemporary research into reinforcement learning applications for UAV control tends to focus on improving the performance of specific higher level trajectory control tasks through the use of more exotic RL algorithms.

## 2.4 A Note on Terminology

This project represents the intersection of a number of different engineering and computer science topics: control systems, machine learning, robotics and aviation. These different topics utilize their own specific terminology to refer to what are essentially the same things. Throughout this paper, some of these terms are used interchangeably. Generally, there is an attempt to use each term in its native context (so that when referring to the effect of a machine learning algorithm on an object, the object will be referred to using the terminology usual in the machine learning field):

**UAS/UAV/MAV/Aircraft/Airframe/Vehicle/Robot/Plant/Environment** This project's focus is the development of flight control systems for small *Unmanned Aerial Systems (UAS)*. The term 'system' is intended to refer to everything involved in the operation of the solution, including ground stations, maintenance crew and operating procedures. The term *Unmanned Aerial Vehicle (UAV)* refers to the actual flying machine. The small UAVs in which this project is interested are typically referred to as *Micro Air Vehicles (MAVs)*. Alternatively, the *vehicle* proper, excluding any attached payload, may be commonly referred to as the *aircraft* or *airframe*.

An unmanned aerial vehicle can be considered a type of *robot*, and the development of such devices falls into the subject of 'mobile robotics' (this is the classification at the University of Canterbury). In the development of control systems for robotics applications, the device being controlled may be commonly referred to using any of the preceding terms. However, in the more general field of control system design, the device being controlled is typically considered as indistinct from the environment in which the device operates. In such situations, the controller is said to operate on a *plant* (a term originating from industrial process control applications).

Finally, the concept that the device and environment in which it operates are indistinct is taken further in the area of machine learning: a machine learning algorithm takes inputs from some *environment*, and performs actions on this environment, to which the environment will respond with some behaviour.

For the purposes of this project, all of the italicized terms may be taken as synonyms; they each refer to the device which is being controlled; in our case, a simulated MAV; that the latter terms also incorporate the larger environment in which the vehicle finds itself has no significant effect.

**Autopilot/Controller/Agent/Policy/Behaviour** Similarly, a range of terminology may be used to refer to the abstract device which actually performs the control of the MAV: the term *flight controller* is commonly used interchangeably with the term *autopilot* to describe the hardware and software which implements the control algorithm. More generally, the field of control engineering would refer to this simply as the *controller*. In the field of machine learning however, this device would typically be referred to as an *agent*. The agent selects actions to perform on the environment to achieve the desired outcome. This can be complicated by considering the agent to be composed of two complementary components: the *learner*, which monitors the behaviour of the environment to determine how it reacts to actions the agent performs; and the controller proper (or *policy*, see below), which chooses which action to perform based on what the learner has learnt so far.

The pattern of actions which the agent chooses as a function of the state of the environment is generally referred to as the agent's *policy*, or less strictly its *behaviour*. This does not really have any direct analogue in traditional control engineering, since for any given (deterministic) algorithm, the behaviour of that algorithm is invariant.

## Chapter 3

# Design of Micro Aerial Vehicles

In this chapter, the general nature and taxonomy of Micro Aerial Vehicles is provided. Then consideration is given to the dynamics of the type of MAV which will be the focus of this project. Finally, a discussion on the design of an available MAV test-stand, and the effect this will have on this project.

### 3.1 About Quadrotor MAVs

#### 3.1.1 General Scheme

Generally, MAVs fall into one of three broad categories, depending on how they source lift to remain airborne: fixed wing aircraft, rotary wing aircraft, and light-than-air aircraft. Further, the rotary wing aircraft can be (roughly) divided into two categories: traditional helicopters, which feature a large single rotor which provides lift, and a smaller perpendicular ‘tail-rotor’ for torque balancing and yaw control; and multicopters, which distribute the generation of lift between multiple rotors, with yaw control delivered by adjusting net torque generated by pairs of counter-rotating rotors. Additionally, a third (fairly recent) category might be single rotor ‘ducts’ which commonly use vanes in the rotor air-stream for manoeuvring (see the Honeywell RQ-16 [15] or Johnson’s analysis [37]). In this project, the focus is specifically on multirotor MAVs.

The multirotor configuration yields a number of advantages over a more traditional helicopter configuration when used in a MAV application: reduced cost (due to the availability of cheap hobbyist electronics and model parts), failure tolerance, and high levels of available control authority being obvious examples.

The most common configuration of multirotor MAV is the ‘quadrotor’; a MAV with four rotors, evenly spaced around the perimeter of a fuselage, all operating perpendicular to the ground and in the same plane. Many variations are possible. The quadrotor has the advantage of requiring the least number of component parts<sup>1</sup>, and allowing simple schemes for coordinating control signals to individual motors. Whilst there is little difference from a control systems perspective (only in terms of motor mixing), in this project, we will concentrate on quadrotor MAVs.

---

<sup>1</sup>Arguably; whilst a triple-rotor MAV has fewer rotors, it must also mechanically tilt one of the rotors to allow yaw control, which is mechanically more complicated than an equivalent quadrotor.

The vast majority of multirotor MAVs use fixed-pitch propellers; the thrust (and torque) generated by each propeller is varied by changing the rotational velocity of the propeller. Where each propeller is directly powered by an individual electric motor, this results in a very reliable system with extremely few moving parts. The ability to rapidly change the rotational speed of each propeller directly affects the available control authority, and hence the control performance of the avionics responsible for flying the vehicle. Because the moment of inertia of the propeller increases with the square of propeller radius, there tends to be some upper bound to propeller sizes where adequate control authority is still available; this tends to limit use of fixed-pitch propeller configuration to MAVs rather than any larger aircraft. Larger aircraft generally use a variable-pitch propeller scheme; in this project we will concentrate on fixed-pitch MAVs.

### 3.1.2 Basic Dynamics

A quadrotor MAV has no lifting surfaces (other than rotors), and all control authority is derived from differential variation in thrust generated by the rotors. Typically, an adequate understanding of the dynamics of the vehicle can be obtained using simple approximations. In general, the dynamics will be governed by Newton's second law,  $F = m \cdot a$  (and the angular equivalent,  $\tau = J \cdot \alpha$ ); to determine the dynamics of the vehicle, approximations of its inertial characteristics  $m$  and  $J$ , and any involved loads  $F$  and  $\tau$  will be required.

Beard (2008) [23] uses a simplified model of the inertial characteristics for a quadrotor, assuming a uniformly dense spherical fuselage with mass  $M$  and radius  $R$ , plus four identical point masses  $m$  rigidly located a distance  $l$  from the centre of the fuselage, to represent motors. Moment of inertia  $J$  is calculated based on this distribution of mass. Hoffmann et al (2007) [34] use an even simpler approximation: the entire vehicle is treated as a point mass  $m$ , with moment of inertia  $I$  to be determined empirically. For this project, the latter approach is selected, since the absolute values of these characteristics should not be important in a learning environment.

Both Beard and Hoffmann et al use a similar approximation for the loads applied to the vehicle: a single gravity force  $F_g$  acting vertically through the centre of gravity of the vehicle; plus a force  $F_n$  and moment  $M_n$  generated by each of the motors  $n = 1 \dots 4$  (as shown in Figure 3.1). Hoffman allows for 'blade flapping' effects, which causes the force  $F_n$  to act non-orthogonally to the rotor plane. Additionally, Hoffmann et al adds a single drag force  $F_d$  which acts through the centre of the vehicle (whilst Beard assumes there are no aerodynamic forces acting on the vehicle). For this project, a similar approach to Hoffmann et al is used; 'blade flapping' is ignored, so each motor force acts perpendicular to the rotor plane; and a damping moment  $M_d$  is added to represent friction effects in rotational motion of the airframe.

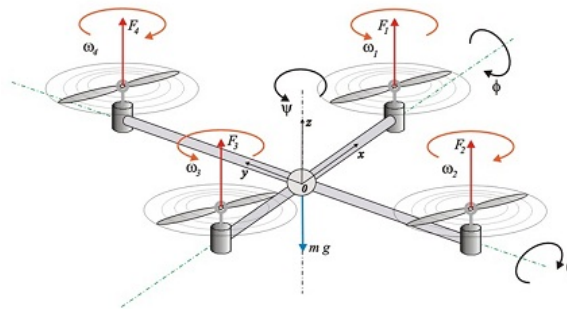


Figure 3.1: Illustrative diagram of simplified model of forces on quadrotor MAV.



In level hovering flight, the four motors of the quadrotor are each required to generate force given by  $F_n = \frac{F_g}{4}$ , to balance net forces on the vehicle. To accelerate upwards whilst level, the force generated by each motor is increased. To accelerate horizontally, the vehicle must first be tilted in the desired direction of travel, then the force generated by each motor increased, so as to obtain a resultant force vector in the desired direction of travel (see Figure 3.2).

Rotation of the vehicle is obtained by differential variation of the force generated by the motors (or the reaction torque, in the case of yaw rotation), so as to produce a net moment acting on the vehicle. Differential variation of thrust ensures that net force generated remains constant, so that rotation may be performed without altering the altitude of the vehicle. However, clearly rotating the vehicle *will* result in changes to the horizontal velocity of the vehicle.

This project will concentrate on the rotational behaviour of the MAV (see §1.2.3), so the rotational dynamics are of most interest. The MAV's rotational dynamics can generally be considered as a second order mass/damper system [30].

### 3.1.3 A Typical Quadrotor UAV

A typical quadrotor MAV is shown as Figure 3.3: the Droidworx CX-4 uses four brushless DC electric motors, mounted on fixed carbon-fibre booms, driving plastic fixed pitch propellers; the central fuselage contains control avionics and batteries. Below the fuselage are mounted landing gear, plus a payload consisting of a thermal camera on a stabilized gimbal.

This particular example is intended for use in Search and Rescue applications. The nature of this task requires operating in challenging environments, such as inside damaged buildings, or in adverse weather conditions. It is for these sort of applications that improvements in flight control systems are clearly valuable.

### 3.1.4 Test Stand

Because multirotor MAVs require a suitable controller to stabilize them, initial testing during the development of such controllers can be difficult. The high risk of damage to the airframe in the event that the controller malfunctions results in a 'chicken or egg' situation; the controller must be working, before it can be tested on the airframe, but the controller needs to be tested in order to know whether or not it *is* working.

In order to address these issues, some method which allows a controller to be tested, without risking damage to the airframe is desirable. One possible solution is to use software simulation of the airframe, rather than testing on physical hardware. This is the solution used for the majority of this project. However, testing only in simulation has its own associated issues, and cannot completely replace testing on the physical plant which is to be stabilized. Hence, the usefulness of a mechanism for testing which acts as a middle ground between software simulation and simply 'setting the controller loose' on a real airframe.

In accordance with these considerations, a 'UAV Test Stand' has been developed [33], and can be used to validate the operation of a prototype control system, without risking serious damage to the airframe. The test stand allows the motion of an airframe to be constrained within fixed boundaries. This prevents the airframe colliding with other objects if the controller fails. Additionally, the test stand allows individual axes to be locked in place, essentially reducing the degrees of freedom of the plant upon which the controller is acting. This permits for UAV

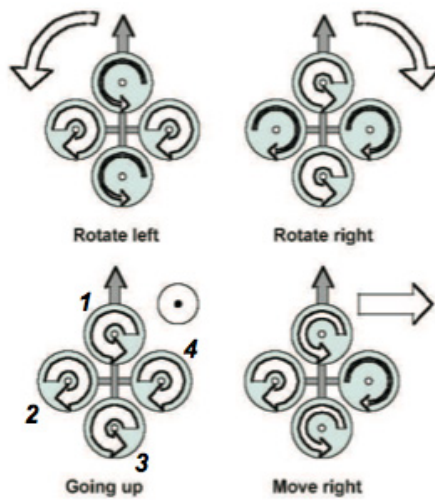


Figure 3.2: Illustrative diagram of principles for movement of a quadrotor MAV.



Figure 3.3: The Droidworx CX-4, a fairly typical quadrotor MAV intended for industrial applications.

control systems to be developed for simple (low degree of freedom) cases initially, then to be extended to more complex cases once the initial concept has been validated.

The current prototype test stand consists of a wheeled two-link arm, as shown in Figure 3.4. This allows for translational movement in the X & Y axes. Translational movement in the Z axis is implemented through the use of vertical draw slides. Finally, rotational freedom is allowed through the use of a three axis gimbal frame. The test stand is designed to minimize any effect on the dynamics of the attached airframe, whilst remaining sufficiently strong to arrest the movement of the airframe when the edges of the allowable working volume are encountered.

In addition to its primary function of restricting the motion of the airframe during controller tests, the test stand is also instrumented with high accuracy sensors, enabling ground truth data regarding the position and orientation of the airframe to be recorded. This ground truth data can then be compared against data recorded by any on-board sensors, so as to permit the characterisation of those sensors.

The sensors used to instrument the test stand are connected via cables to a computer for data logging. These cables place constraints on the rotational movement of the airframe: the airframe is restricted to a maximum of  $\pm 180^\circ$  in yaw (Z axis), and  $\pm 45^\circ$  in each of pitch and roll (X & Y axes). This prevents the cables wrapping around the gimbal. Whilst one expects this to be addressed using wireless sensors in the future, these current limitations were considered when designing the simulator used in this project, so as to maximize compatibility between simulated tests and future tests which may be conducted using hardware on the test stand.

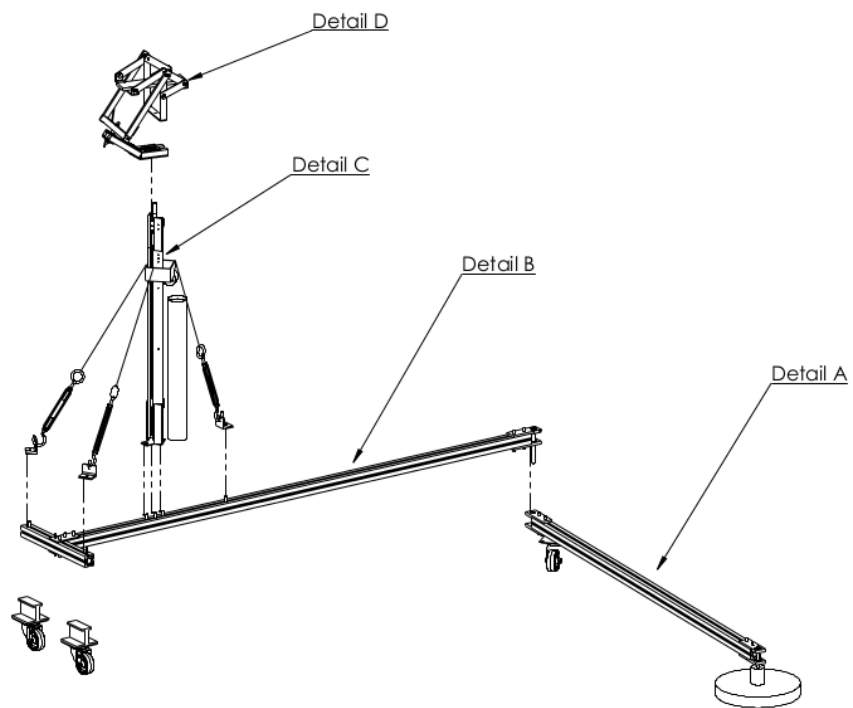


Figure 3.4: The UAV test stand. The X/Y translational arm is denoted Details A and B. The translational Z slide is denoted Detail C, and the rotational gimbal frame is denoted Detail D.

## Chapter 4

# Software Design

In this chapter, the design, functionality and internals of the software needing to be created for this project are discussed in detail: primarily a software simulator of MAV flight dynamics, and a prototype Reinforcement Learning based flight controller. Additionally, the design decisions behind the selection of software tools and libraries to be utilized are explained.

### 4.1 Computing Software Technologies

In the undertaking of any software development, selection of appropriate software technologies to leverage is extremely important. For this project, software technologies were identified based on a number of criteria: performance, compatibility with hardware, ease of integration and quality of documentation. Accordingly, a number of software technologies were identified as being suitable for use with this project:

#### 4.1.1 Java

##### 4.1.1.1 Overview

The Java language was selected as the basis for the software development. Java is a class-based, object-oriented programming language, whose syntax is closely based on C/C++ [15] [11]. Java is designed for portability, and is intended to be compiled into machine portable byte-code which is run on a virtual machine. The virtual machine typically uses a just-in-time compiler to internally convert byte-code into native machine code, reducing performance issues due to interpretation. The absence of low level constructs such as pointers, and the use of automatic memory management, means the developer can concentrate on developing application code rather than underlying functionality.

Java was originally developed by Sun Microsystems (now Oracle) [12], intended as a component of the complete ‘Java Platform’ which was a mechanism for deploying application software cross-platform, including across embedded or internet based environments. However, Java has gone on to become one of the most popular programming languages in the world [7] [28] [18], including usage far outside its original remit.

#### 4.1.1.2 Suitability

Java has excellent documentation, with support widely available online. It is easily portable between the hardware platforms expected to be involved in the project; whilst the requirement for a virtual machine limits the ability to run Java on many embedded platforms, the complex algorithms used in this project are likely to only be viable on embedded systems which are capable of running Linux, in which case a Java Virtual Machine will most likely be readily available. Additionally, there are a great number of third-party libraries available for Java, and integrating such libraries into a Java software project is simple. This makes Java a suitable choice for use as the starting point for software development in this project.

The most glaring weakness of Java is its speed. Being compiled to byte-code and executed through a virtual machine is never going to be as optimal as compilation to native machine code which can be executed directly. However, modern Java implementations have vastly improved performance over the originals; this is primarily through the use of just-in-time compilation to improve runtime performance (though at the cost of greater initialization time). Performance of Java currently seems to be comparable to that of natively compiled languages such as C for commonly used software tasks, though there are still differences in particular applications [38] [32] [27]. Performance is of great significance in terms of evaluating the viability of candidate control algorithms (see §7). However, considering that the software will be performing simulations which execute many iterations of the same code, the benefits of just-in-time compilation will be maximised, and thus the performance of Java should be sufficiently close to that of C/C++ as to be acceptable for this project.

#### 4.1.1.3 Alternatives

The clearest alternatives to Java would be C or C++ (most likely the latter). Both C and C++ are essentially ubiquitous, especially in terms of embedded software development [15] [18]. C/C++ offers the maximum possible in terms of performance (short of hand-coded assembler), largely due to the maturity of C compilers, and the plethora of low level functionality which C/C++ programmes can utilize. However, this low level approach makes C/C++ significantly more complicated, especially in terms of memory management, incorporation of external libraries and integration with other software.

Another possibility would be Python, a portable multi-paradigm language originally intended as a scripting language [16]. Python offers portability and high-level functionality such as dynamic typing and memory management, which would confer many of the same benefits as Java in terms of development. However, the official Python implementation does not use just-in-time compilation, instead interpreting byte-code directly by a virtual machine. This makes Python the least preferable of the alternatives, given the importance of runtime performance in the context of this project.

### 4.1.2 Reinforcement Learning Library (Teachingbox)

#### 4.1.2.1 Overview

Teachingbox is a Java library which provides an easy method for developers to integrate machine learning technologies with robotic systems. The library is open-source (hosted on SourceForge),

but is primarily developed by the Collaborative Centre for Applied Research on Service Robotics, based at the Ravensburg-Weingarten University of Applied Sciences (Germany) [8].

Teachingbox provides implementations of a variety of reinforcement learning algorithms, in addition to considerable support infrastructure to allow simple integration of a chosen algorithm into an application specific environment, and also functionality to support testing and perform diagnostics.

#### 4.1.2.2 Suitability

The software has the advantage of being written in Java, making it easy to integrate with the other software developed during the course of this project. The library is distributed with sufficient, though not comprehensive, documentation permitting it to be used without difficulty.

Teachingbox provides actual implementations of a number of machine learning algorithms, in addition to the infrastructure to integrate the algorithms. This is a significant benefit, as it alleviates the need to develop implementations of known reinforcement learning algorithms specifically for this project. This will substantially reduce development time.

The only concern is a lack of developer activity; no new releases have been made since 2010, and there is no activity on the support pages for the software since 2011. However, since all the functionality required from the library for this project has already been implemented, the software remains suitable. Consideration should be given to alternatives if additional development, beyond the scope of this project, is undertaken.

#### 4.1.2.3 Alternatives

A possible alternative to the functionality provided by the Teachingbox library is RL-Glue [5] [51]. The RL-Glue project provides a standard interface for interconnection between different reinforcement learning components. The system is intended to operate with components written in different languages, and uses a server/socket based architecture to achieve this; binding packages (referred to as ‘codecs’) which provide support for a specific language are available for C/C++, Java, Lisp, MATLAB and Python among others.

Whilst RL-Glue is perhaps in more widespread usage, it has disadvantages compared with the Teachingbox library: most significantly, while RL-Glue provides the necessary infrastructure for connecting reinforcement learning components, it does not provide implementations of the algorithms themselves. This would mean that the desired algorithms would need to be sourced separately, or programmed from scratch.

### 4.1.3 J3D Robot Simulator (Simbad)

#### 4.1.3.1 Overview

Simbad is a Java library which implements a 3D robotics simulator [36]. It is mainly intended for research or teaching involving 2D ground based robotic agents, as a method for testing artificial

intelligence algorithms [35]. The library is open-source (hosted at SourceForge) and depends upon the common Java3D library for rendering.

The tool-kit allows for the simulation of multiple robots, and implements a variety of simulated sensors which can be used inside the simulation environment. The library also provides support for simulation of collision interactions between simulated objects, but this functionality seems immature.

#### 4.1.3.2 Suitability

Similarly to the Teachingbox library, Simbad is written in Java (making it easy to integrate with the other software components) and is distributed packaged with sufficient (though also not comprehensive) documentation.

Although the simulation functionality offered by Simbad is not ideal for the work performed as part of this project, it contains an excellent basis for modification. This alleviates the need to write a complete simulator package from scratch, which will substantially reduce development time.

Development on Simbad ceased as of 2007. However, as of 2011 support remained available.

The most significant downside to the use of Simbad as a simulator library is the design of some of the object interfaces which the library presents. A number of important fields are declared ‘private’ rather than the less restrictive ‘protected’, which means these fields are inaccessible when extending the object. This makes some inelegant duplication necessary, though it should not significantly affect the performance of the software.

#### 4.1.3.3 Alternatives

A number of other Java based robotics simulators exist, and could plausibly be suitable for this project. The best candidate would be MASON, a multi-agent 2D and 3D simulation library, similar to Simbad. MASON, being more comprehensive than Simbad, is thus substantially more complicated. It may be an appropriate choice if a more detailed simulation was called for in the future [1] [22].

Another approach would be to use Gazebo, a 3D multi-robot simulator which includes simulation of rigid body physics for simulated objects [10]. Gazebo was previously known as part of the open-source Player Project (commonly referred to as Player/Stage) [4]. As with the Player Project, Gazebo is written in C++, but has bindings available for a wide range of languages. Gazebo is more complicated than is required for this project, utilizing a client/server, plug-in-capable architecture, and would make development for this project substantially more difficult.

## 4.2 Quadrotor UAV Simulator

### 4.2.1 Encoding of Simulator State

As with most other 3D robotics applications, a free-flying UAV operates in six degrees of freedom (DoF), three translational and three rotational.



In terms of the simulation, there are two relevant frames of reference against which to measure: the global (or ‘earth’) frame, and the local (‘base’ or ‘vehicle’) frame. The global frame is fixed within the simulation, with the XZ plane representing the ground, and the positive Y axis representing ‘upwards’. The local frame moves around within the global frame, attached to the simulated UAV: the origin of the local frame is always at the centre of the UAV’s body, with the local XZ plane always lying in the plane of the UAV rotors, the X-axis aligned with the ‘front’ of the UAV. Hence, in typical aviation terminology, X-rotation is ‘roll’, Z-rotation is ‘pitch’, and Y-rotation is ‘yaw’. The configuration of the two reference frames, and the transformation from one to the other, is shown in Figure 4.1.

The dynamics of the simulated UAV must be evaluated in terms of the local frame. For instance, when power to the rotors is increased, the UAV accelerates along the *local* Y-axis. However, the position of the UAV must be measured in the global frame (since the UAV is *always* at the origin of the local frame). Thus, having evaluated the dynamics of the UAV in the local frame, the velocity in the local frame must be transformed into the global frame to allow the position of the UAV to be updated.

There are multiple possible encoding schemes for the rotational position (orientation) of the UAV in the global frame [40] [29]. The scheme used here is intrinsic Tait-Bryant [15], in the order Z-Y-X (pitch first, yaw, then roll).

#### 4.2.2 Freedom During Testing

Since a free-flying UAV operates in six degrees of freedom, in order to be a viable approach to controlling a UAV, a control system needs to be able to operate in six degrees of freedom. However, a control system will commonly be initially developed and tested on less complex systems to validate it. A common method of achieving this is to fix one or more ‘joints’ of the system under test.

In the case of developing control systems for UAVs at the University of Canterbury, a dedicated UAV ‘test stand’ has been under development within the Mechanical Engineering Department, which allows some axes of motion of a mounted UAV airframe to be fixed at a specified value (or restricted to some range of values), whilst allowing the other axes to move unhindered. Accordingly, the simulated experiments were designed to reflect how physical experiments would be performed.

Within the scope of this project, it was decided to limit experiments to the three rotational degrees of freedom. During each test, the translational velocity of the simulated UAV was set to zero, and information regarding translational position and velocity was discarded. Only information encoding the rotational state of the simulated UAV was passed from the simulator software to the control software. Hence, the six degree of freedom simulation was reduced to three degrees of freedom.

Further, each of the three rotational degrees of freedom was handled by a separate controller; giving three individual controllers, each operating on a single degree of rotational freedom. The assumption being made is that the dynamics of each degree of freedom are entirely uncoupled. This is known to be false: the dynamics of each rotational axis are coupled with one another. For example, the Z-Y-X encoding scheme used for rotational position means that if the UAV pitches up (Z-rotation) by ninety degrees, rolls ninety degrees (X-rotation), and then pitches back down again, the final orientation of the UAV will be encoded as a ninety degree yaw angle, even though no explicit Y-rotation occurred. However, it was assumed that for low angles of rotation, the coupling would be sufficiently small that the controllers would still operate satisfactorily.

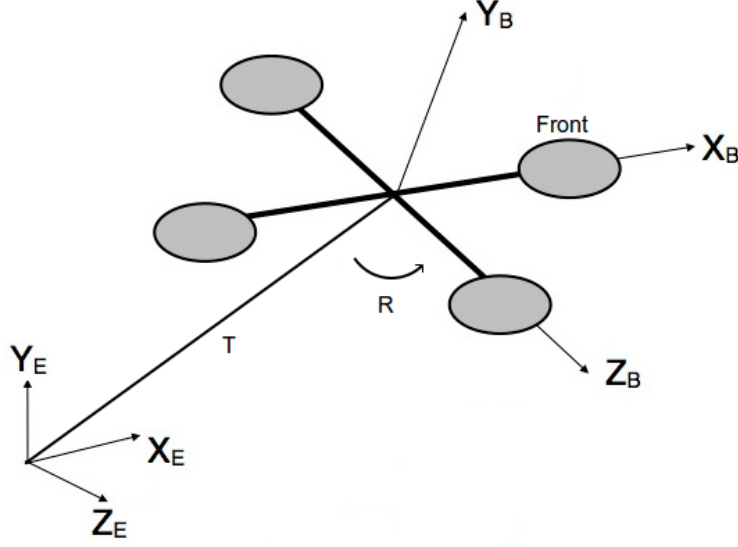


Figure 4.1: Diagram showing the global/earth reference frame (denoted  $E$ ) and the base/vehicle reference frame (denoted  $B$ ). The transformation from frame  $E$  to frame  $B$  is composed of a translational component (denoted  $T$ ) and a rotational component (denoted  $R$ ).

Additionally, the range of motion allowed for each axis was restricted to match the range of motion expected during physical experiments using the UAV test stand [33]. The Y-axis rotation was restricted to a maximum of 180 degrees either side of the origin, whilst X & Z rotation was restricted to a maximum of 45 degrees either side of the origin. It was anticipated that this would not be a serious restriction on the validity of experimental results, since any rotation in the X & Z axes greater than 45 degrees is uncommon in practise: because large pitch and roll angles reduce the vertical thrust component the airframe generates, they can result in rapid loss of altitude and a crash.

#### 4.2.3 Encoding of Controller State

State information from the simulator components must be converted for use in the controller components of the software, specifically into a form compatible with the Teachingbox library. The Teachingbox API defines two classes, **State** and **Action**, which encode information regarding the state of the environment and the selected action for the agent to perform. Both of these classes are essentially wrappers for a single dimensional array of double precision floats.

Because independent controllers are being used for each of the rotational axes, each **State** object includes only information which determines the state of the simulated quadrotor in a particular axis. To fully define the state of a quadrotor about a single rotational axis, two characteristics are required: the angular position (displacement) relative to some origin, and the current angular velocity. Hence, the **State** for each controller is composed as shown in Equation 4.1.

$$S_i = [ \theta_i \quad \dot{\theta}_i ] \quad (4.1)$$

Where  $S_i$  denotes the encoded state for axis  $i$ ,  $\theta_i$  denotes the angular displacement of the quadrotor about axis  $i$  (measured in radians), and  $\dot{\theta}_i$  denotes the angular velocity about axis  $i$

(measured in radians per simulator time step).

Because independent controllers are being used for each of the rotational axes, each **Action** object includes only information about how the simulated quadrotor should assert its control authority in a particular axis (these signals are later mixed together, see §4.2.4). A single characteristic is required: the rotational torque to apply. Hence, the **Action** output from each controller is composed as shown in Equation 4.2.

$$A_i = [\tau_i] \quad (4.2)$$

Where  $A_i$  denotes the encoded action for axis  $i$ , and  $\tau_i$  denotes the desired torque to be applied about axis  $i$  at the next time step.

The Teachingbox library offers support for continuous states, but only for discrete actions. The observed values of the environment are not discretised before being passed into the selected policy (although the policy may be using an algorithm which discretises the states internally). On the other hand, the policy must select from a predetermined set of possible actions (provided to the **Policy** class in the form of an **ActionSet** object, which gives a set of all the permissible actions).

Increasing the number of possible actions greatly increases the number of iterations required to learn a policy which selects amongst these actions. To reduce the complexity of the policy being learned, a minimal set of two actions was selected; each controller either asserts full authority in one direction or the other (‘bang-bang’ control). For example, the **ActionSet** for the X (roll) axis has two elements: an **Action** called **ROLL\_LEFT** and an **Action** called **ROLL\_RIGHT**.

The range of possible discrete actions is represented through an **ActionSet** object. This encapsulates a list of all possible actions. It may also include an **ActionFilter**, which can be used to impose further restrictions, such that only some actions are permitted when in a particular **State**. For this project, no **ActionFilter** was used, so the complete set of actions is available to the agent, regardless of which **State** the robot is in.

Support for continuous Actions may be addressed in the Teachingbox library in the future.

#### 4.2.4 Throttle Signal Mixing

Because independent controllers are being used for each of the rotational axes, the output action from each channel of the controller represents the desired corrective action for that specific axis. In a conventional fixed wing aircraft, there are independent control surfaces for each axis, so the output from each channel of the controller may be routed directly to the appropriate control surface. However, in a multirotor vehicle, manoeuvring in all six degrees of freedom is achieved by coordinated variation of all motor speeds. This means that the independent axis control signals from each controller need to be mixed together, in order to obtain coordinated commands to be output to the vehicle’s motors.

In this software, mixing of control signals is performed by the **QuadMotorMixer** class. The class’ static **mix()** method is called from the **QuadSimbadAgent** class at each simulator time step. Individual command signals from the controller for each axis of the experiment are converted into coordinated motor command signals which have the desired affect on the motion of the simulated vehicle.

The mixing algorithm takes as input a command signal for each of the roll, pitch, and yaw axes. Additionally, it takes a throttle signal, though for the non-translational case considered in this project, this is fixed at 50% of full power. Each of the input command signals should be a percentage value, from -100% to +100%, which represents the desired control action in that particular axis. The following algorithm is then used to calculate the required motor power levels:

$$\begin{aligned}
P_0 &= T + A_{pitch} + A_{yaw} \\
P_1 &= T - A_{roll} - A_{yaw} \\
P_2 &= T - A_{pitch} + A_{yaw} \\
P_3 &= T + A_{roll} - A_{yaw}
\end{aligned} \tag{4.3}$$

Where  $P_0 \dots P_3$  are the power levels for the front motor, right motor, rear motor and left motor respectively;  $A_{roll}$ ,  $A_{pitch}$ ,  $A_{yaw}$  are the percentage command signals for each of the rotational axes; and where  $T$  is the throttle signal, fixed at 50.

If any of the resulting values are negative, they are clipped to a minimum of zero. Finally, if any of the resulting values are greater than 100, then *all* of the resulting values are scaled down proportionately, such that the maximum value is 100. The values now represent the percentage of full power which should be applied to each motor.

Other methods for scaling the outputs from the throttle mixing are equally possible. This method was selected, as it prevents a large control action in one axis from saturating the motor outputs, which could prevent control action in other axes from being represented in the output motor coordination.

#### 4.2.5 Simulator Implementation

The behaviour of the quadrotor simulation is implemented primarily through two classes in the `simbad.sim` package: `QuadSimbadAgent` and `QuadSimbadKinematicModel`.

##### 4.2.5.1 Class QuadSimbadAgent

The class `QuadSimbadAgent` implements the simulated 3D body of the UAV within the Simbad simulator environment, and additionally provides implementation of high level behaviour of the simulated UAV. The agent controls the configuration and position of the simulated UAV within the simulator environment, using the `QuadSimbadKinematicsModel` class (see §4.2.5.2) to calculate the UAV's dynamics.

The `QuadSimbadAgent` class is instantiated by providing it with references to the control systems to be used, using the `SimbadRobotController` type. When the class is instantiated, the Agent class creates an instance of the `QuadSimbadKinematicsModel` which it intends to use, and attaches this to itself. Additionally, it creates an instance of each of two private classes, `QuadIMUSensor` and `QuadMotorActuator`, which are also attached to the Agent. The use of these private classes provides separation of the abstractions of sensors and actuators for the simulated quadrotor. This helps to ensure as realistic a simulation as possible, and allows for testing the effects of changing sensors or actuator characteristics on controller performance with only minimal changes to the simulator. Finally, the Agent allocates the three provided `SimbadRobotController` objects to each of the axes to be controlled.

While the simulator interface is open, the simulator uses the `create3D()` method of the `Agent` to update the 3D position of the simulated UAV within the simulated environment. The `create3D()` method draws a stylized depiction of a quadrotor UAV consisting of a rectangular ‘body’, and four cylindrical ‘motors’, with a small indicator to differentiate the ‘front’ of the vehicle, as shown in Figure 4.2. Since the UAV is being simulated as a rigid body with arbitrarily chosen mass and moment of inertia, and since the UAV is operating in an otherwise empty region free of objects with which to collide, there is no need to simulate the UAV with objects of any particular shape; an extremely small, purely spherical ‘body’ would be just as effective. However, the Java3D library on which the Simbad simulator is based does not differentiate between simulating a 3D scene and rendering the scene so it can be visualised by the user. Thus a UAV is simulated which looks similar to a real quadrotor, enabling the user to see what is happening during each experiment.

When the simulator is running, at each time step, the simulator calls the `updatePosition()` method of the `Agent` class. The `updatePosition()` method first checks to see whether the rotational position of the UAV means it has come into contact with the boundaries of the working volume (as per §4.2.2). If the UAV has come into contact with the boundaries of the working volume, the velocity of the simulated UAV is set to zero, and its position is set to just inside the offending boundary. Following this, the position of the simulated 3D objects associated with the UAV is updated by moving the UAV some incremental displacement, as calculated by the `QuadSimbadKinematicsModel`.

Additionally, whilst running, at each time step, the simulator calls the `performBehaviour()` method of the `Agent` class. This method is set aside for implementing the intelligence under test, to separate this behaviour from that associated with maintaining and updating the 3D simulation itself. In the case of the `QuadSimbadAgent`, the `performBehaviour()` method first uses the `getRobotState()` method, which interrogates the private `QuadIMUSensor` object to obtain information regarding the state of the UAV. This data is passed to the `performBehaviour()` methods of the individual `SimbadRobotController` associated with each of the rotational axes. The individual controllers return `Action` objects, which are mixed according to the throttle mixing scheme (see §4.2.4) to obtain appropriate power levels for each of the UAV’s motors. These power levels are then specified to the private `QuadMotorActuator` object, which changes the power levels used by the `QuadSimbadKinematicModel` object in the next simulator iteration.

#### 4.2.5.2 Class `QuadSimbadKinematicModel`

The class `QuadSimbadKinematicModel` implements the dynamic behaviour of the simulated UAV. The `Agent` indicates to the kinematics class what percentage of ‘full power’ should be applied to each of the UAV’s motors through the `setPropPower()` method. The physical state of the modelled `Agent` is then simulated by calling the `update()` method once every simulation time-step.

The `update()` method takes the amount of time elapsed since the previous update, and current rotational position of the agent. The current translational position is *not* required, since (assuming the translational space is boundless and free of obstruction) knowing the translation position of the agent is not necessary for simulating either the rotational or translational dynamics. The method essentially returns the instantaneous changes in translation and rotation which the simulator environment should apply to the 3D model of the agent so as to represent its simulated behaviour during the next time step. The class retains its own record of the translational and rotational velocity of the agent.

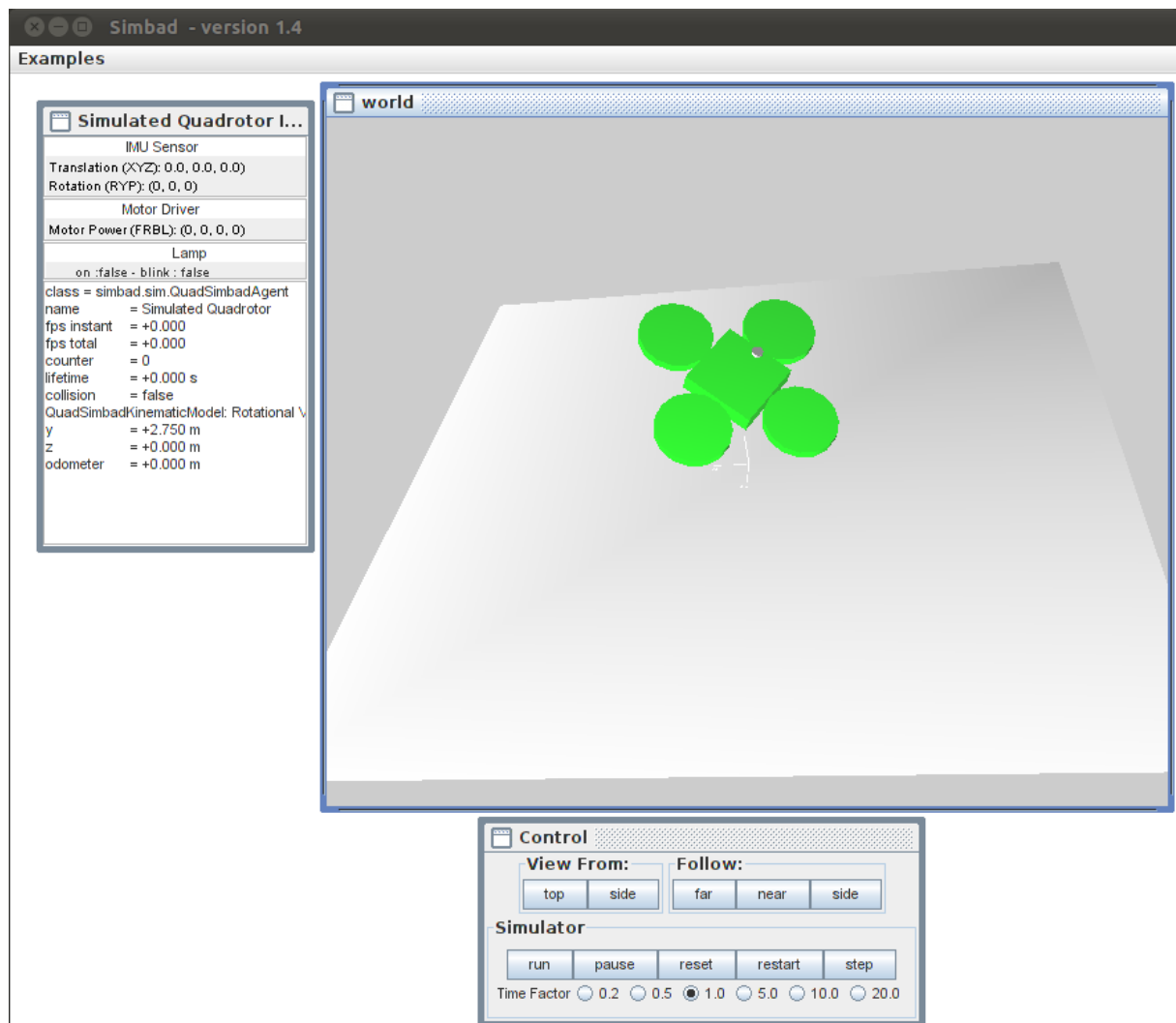


Figure 4.2: Screen-grab of simulator user interface, showing visualisation of simulated UAV.

At each time step, the simulation is updated as follows: Firstly, the translational velocity of the agent is updated. The total force applied by the propellers, normal to the plane of the propellers, is calculated based on the current propeller power levels. Based on the rotational position of the agent, the component of this total force acting in each of the X, Y and Z axes is calculated. Friction forces are added to these forces, calculated based on the current translational velocity in each axis, plus gravitational force on the Z axis. The total force acting in each axis gives the instantaneous translational acceleration in that axis. The translational velocity is then updated by integrating the instantaneous acceleration over the step time. Secondly, the rotational velocity is updated. The moment applied to each of the local axes (Roll, Pitch, Yaw) is calculated based on the differences in applied propeller power levels. To each moment, friction moments are added, calculated based on the current rotational velocity. Random moments to simulate disturbances may also be added. The total moment on each axis gives the angular acceleration around that axis. The angular velocity is then updated by integrating the instantaneous acceleration of the step time. Finally, the required instantaneous changes in translation and rotation are determined by integrating the calculated velocities over the step time. Note that because of the design of the Simbad tool-kit, rotation data is expressed by the kinematics class with respect to the local frame of the agent being simulated (rather than rotating it back into the global frame).

In a discrete time system such as a simulator, high rotational velocities can result in unusual behaviour due to aliasing if the angular velocity becomes so high that one complete revolution (or more) is completed within a single simulator time step. To address issues arising from this situation, the kinematics class imposes some hard limits on the angular velocity attainable by the agent. For this project, the maximum angular velocity was specified such that the agent must take no less than sixty simulator time steps to cross from one side of the working volume to the other. A more permanent solution might involve modifying the simulator to prevent problems due to aliasing, but this was outside the scope of the project.

Because in this project, only a single agent is simulated, in an otherwise empty region, there is no need to incorporate additional complexity to deal with handling collisions between the agent and other simulated objects. Whilst the Simbad tool-kit does offer facility for collision detection, allowing for realistic collisions between simulated objects, the implementation was found to be overly complex for the scope of this project, and so was disabled. Since there was no need to simulate complex interactions between moving objects, no further efforts were made to use the built in collision detection.

The only collision handling complexity required, is interaction between the simulated agent and the ‘operating limits’ for each degree of freedom (see §4.2.2). Perfectly inelastic collisions with the limits of each axis are simulated by the **Agent** class calling a method which sets the velocity in this axis to zero (for instance, the method `zeroTransVelX()` would set the translational X-axis velocity to zero).

## 4.3 Controllers

Within the scope of this project, two different control algorithms were utilized. Each control algorithm is implemented within a Teachingbox **Agent** type object, which encapsulates a **Policy** object which maps **States** (as observed from the environment) into **Actions** to be taken by the control system.

### 4.3.1 Traditional PID Controller

To provide a baseline against which the performance of a machine learning control algorithm may be compared, a ‘conventional’ control system was implemented, in the form of the common ‘PID’ (Proportional, Integral, and Derivative) controller.

In the PID case, the **Agent** encapsulates a policy of the **PIDPolicy** class. The **PIDPolicy** class implements a traditional PID controller which operates on a particular scalar; when the class is created, an index must be specified which selects a particular entry out of the input (vector based) **State** for the controller to act upon. In this case, the selection is the angular displacement of the vehicle.

As expected, the controller first calculates the error between the nominal value, and the current input value. This error is used to estimate the integral of the error (by adding the current error to the previous estimate of the integral), and derivative of the error (by subtracting the previous value of the error from the current error). Note that, in this project, the derivative is calculated independently of the angular velocity, which is present in the input **State** but unused.

No consideration is given to preventing integral wind-up, nor to filtering derivative values.

The appropriate controller action is then calculated by multiplying the error, error integral, and error derivative by a corresponding gain value, and summing the results (as shown in Equation 4.4).

$$C_n = E_n \cdot K_p + \left( \sum_{i=0}^n E_n \right) \cdot K_i + (E_n - E_{n-1}) \cdot K_d \quad (4.4)$$

Where  $C_n$  is the calculated output action for the  $n_{th}$  time step,  $E_n$  is the input scalar value for the  $n_{th}$  time step, and where  $K_p$ ,  $K_i$  and  $K_d$  are the chosen proportional, integral and derivative gains (respectively).

Since within the scope of this study there is no need to modify the controller gains at runtime, the gains are implemented as constant values.

#### 4.3.1.1 Selection of Action

The PID control algorithm outputs a numerical value; this needs to be transformed into one of the available set of actions present in the **ActionSet** for the environment. To do this, the **PIDPolicy** iterates through each of the actions in the **ActionSet**, and selects the **Action** which is closest to the calculated numerical action value (for a specific index within the **Action**, in the case that the **Action** is a vector).

For this project, where there are only ever two entries in the **ActionSet** (as per §4.2.3), the controller will select one of the available actions if the calculated output is a positive value, or the other **Action** if the calculated output is a negative value. The **Action** selected in this manner is then output from the controller to the simulator stages.

This means that the controller will, in fact, be acting as a ‘bang-bang’ controller rather than a true PID controller. However, the Q-Learning controller will also only select between discrete states (see §4.3.2.4). Further, since minimum-time optimal control strategies typically involve



saturated controller outputs during the majority of a step response (as discussed by Sonneborn [48]), it is expected that the performance of the bang-bang controller will only differ significantly from the PID controller for small error values. Hence, we assume that the observed comparative performance between controllers in the ‘two action’ configuration will be indicative of the results when the number of available actions is increased (which tends towards a true PID controller); the extension of testing to multiple actions is discussed in §8.3.6.

#### 4.3.1.2 Tunable Parameters

As with any basic PID controller, there are three important parameters which need to be tuned to obtain optimal performance from each controller. As a PID controller does not exhibit any learning behaviour, the performance of the controller is entirely determined by these parameters.

The tunable parameters are  $K_p$ ,  $K_i$ , and  $K_d$ ; the gains for the proportional, integral and derivative terms of the controller (respectively). For sophisticated variants of a PID controller, there may be additional parameters to configure functions such as integral unwinding. However, for this study, these are not used.

### 4.3.2 Q-Learning Controller

#### 4.3.2.1 Reinforcement Learning System Design

As introduced in §2.1, characterising the nature of the task  $T$ , the performance measure  $P$  and experience  $E$  is the first step in the design of a machine learning system. Accordingly, consideration is given to how these parameters are represented in the domain of this project.

The task  $T$  is to move the robot from an initial state into a known nominal state; the specific manner in which the task arranged is detailed in §6.1.1. The performance measure  $P$  is the number of iterations required to settle the robot at the nominal state; the specifics of this metric are discussed in §6.1.2. Finally, the experience  $E$  consists of the previous attempts by the system at controlling the robot. The self-referential nature of experience  $E$  defines this as a reinforcement learning approach.

The next phase of design for a machine learning system entails selection of a target function, the manner in which this target function is represented, and an algorithm which will be used to approximate the function. After considering a number of possible reinforcement learning algorithms (see §2.3 and the Teachingbox documentation [8]), it was decided that efforts would be focused on the Q-learning (QL) algorithm, using an Adaptive Radial-Basis Function Network (ARBFN) as a function approximator for the Q-function. The target function is hence the Q-function, which is itself driven by the reward function; the specification of which is described in §4.3.2.3.

#### 4.3.2.2 System Implementation

The implementation of the Q-learning algorithm used from the Teachingbox library actually provides an implementation of  $Q(\lambda)$  (with an unspecified scheme for handling exploratory

actions) utilising eligibility traces (see §2.2.3). An  $\epsilon$ -greedy method for action selection was chosen, to ensure thorough exploration of the state/action space for each controller.

To implement this, the **Agent** encapsulates both a policy of the **EpsilonGreedyPolicy** class, and a learner of the **GradientDescentQLearner** class. The learner side gradually creates a Q-function, which the policy side then uses to select the most appropriate **Action** to take, given a specific **State**.

The process to instantiate the controller is as follows: A new **RadialBasisFunction** object (which implements the radial basis function used in the ARBFN) is created, along with a new **RBFDistanceCalculator** object (which determines when a new RBF term needs to be added to a network, based on the ‘density’ of existing terms). These two objects are used to create an instance of the **Network** class, which implements the ARBFN.

The ARBFN, and an **ActionList** enumerating the available actions, are then used to create a Q-function through the **QFeatureFunction** class.

The Q-function is used by the controller’s **Policy** to select the **Action** to perform at each time-step. In this case, the **Policy** used is the **EpsilonGreedyPolicy**. At each time-step, the **EpsilonGreedyPolicy** performs an exploratory (randomly selected) **Action** with probability  $\epsilon$ , or an exploitive **Action** (based on the Q-function) with probability  $(1 - \epsilon)$ . The probability  $\epsilon$  remains constant throughout each experiment, as defined by the **ConstantEpsilon** class.

Additionally, at each time-step, the Q-function is updated by the **Learner** component of the controller. In this case, the **GradientDescentQLearner** class is used; at each time-step, the Q-function local to the current **State/Action** pair is updated by performing gradient descent in the error space, based on the provided reward value.

#### 4.3.2.3 Reward Scheme

The reward scheme for the reinforcement learning based controller is implemented through classes of the **EnvironmentObserver** interface; in this case the classes **QuadEnvObserverX**, **QuadEnvObserverY**, and **QuadEnvObserverZ**. Whilst the individual classes may have different values for some fields, the general form of the reward scheme implemented remains the same for all axes.

At each simulator time-step, the reward value is provided to the **Agent** by the environment; the **Agent** then passes the reward value onto the **Learner** component, along with the current **State/Action** pair to which the reward value matches.

The controller is designed to position the robot at a nominal position; the reward scheme should be designed so that when the robot is located at the nominal position, the reward value will be highest; since Q-learning prefers state/action pairs with higher Q-values, such a reward function *should* result in a policy which moves the robot towards the nominal state.

Accordingly, the reward scheme uses a term which is proportional to the absolute error in the angular displacement of the quadrotor in a specific axis. When the robot is located correctly, the absolute error is zero, as is the reward value. If the robot is not located correctly, the absolute error will be greater than zero, and the reward value will be less than zero. Thus, the reward value will generally be higher if the robot is located correctly.

A reward scheme which only rewards the angular displacement of the robot may result in policies which rapidly slew the quadrotor backwards and forwards *through* the nominal point,

achieving a higher reward value than might be expected. The **State** of the quadrotor having the correct angular displacement, but very high angular velocity, has the same reward value as being stationary at the correct angular displacement; this clearly is not what the designer would intend. To compensate for this, an additional term is added to the reward scheme, which adds a penalty proportional to the square of the angular error velocity in the relevant axis. The square of the velocity is used, so that high speeds are strongly penalized, whilst low speeds are minimally penalized, encouraging the agent to select a policy which minimizes motion of the quadrotor. The general form of the reward scheme is thus shown in Equation 4.5.

$$R = K_{\theta} \cdot |\theta - \theta_n| + K_{\dot{\theta}} \cdot \left| \dot{\theta} - \dot{\theta}_n \right|^2 \quad (4.5)$$

Where  $R$  is the reward value for the current state,  $K_{\theta}$  and  $K_{\dot{\theta}}$  are reward gains for the displacement and velocity terms (respectively),  $\theta$  and  $\dot{\theta}$  are the angular displacement and angular velocity terms (respectively) of the current state (as encoded in the **State** object as per §4.2.3), and  $\theta_n$  and  $\dot{\theta}_n$  are the angular displacement and angular velocity terms (respectively) of the nominal state the controller is trying to attain.

Both  $K_{\theta}$  and  $K_{\dot{\theta}}$  must be negative values, so that the reward scheme favours stationary states, close to the nominal position, over moving states, far from the nominal position.

**Reward Scheme at Limits of Motion** Through the `isTerminal` flag, the environment observer class is able to tell when the simulated robot is constrained through some boundary conditions. In this case, the `isTerminal` flag is set by the `SimbadLearningRobotController` class, according to its `isBounded` flag. This is in turn is set by the `collisionTestStandRotationX` method (and matching methods for the Y & Z axes) in the `QuadSimbadAgent` class. This allows the reward scheme to vary as a function of whether or not the robot has struck the limits of its work envelope.

This is necessary because the manner in which the limits of the work envelope are implemented in the simulator will result in the robot learning to ‘stick’ against the side of the work envelope unless some compensation is provided.

When the robot encounters the edges of the working envelope, the simulator arrests the robot by immediately setting the relevant component velocity to zero. Hence, at the time-step immediately *after* the robot moves away from an edge, it will be located distance  $d$  from the edge, with velocity  $v = d \cdot \delta t$ . The change in reward during this time-step will have a positive component contributed by  $d$ , plus a negative component contributed by  $v$ . For some values of reward gains and step size, the change in reward may be negative (i.e. the agent would prefer to remain stationary at the edges).

The Q-function is initially flat, so a negative gradient of immediate reward at the edges may be enough to generate a Q-function which slopes upward towards the edges of the working volume, forming a ‘lip’ which traps the agent. With sufficient exploration, back propagation means that eventually, the vastly superior reward values near the nominal state will overwhelm the effects of the edges. However, there is a risk that early in the experiment, the agent can become trapped against the edges of the working envelope, thus preventing it from completing the necessary exploration.

The severity of the problem depends on a number of other simulation parameters. For higher resolution Q-functions and smaller step sizes, the upturned ‘lip’ at the edges of the working

volume is reduced in ‘width’, making it less likely for the agent to become trapped. Additionally, if the epsilon value for the experiment is sufficiently high, eventually a sequence of exploratory actions will be taken, moving the agent out of the affected region.

Relying on a high resolution for the Q-function and high epsilon value is an inconvenient method of addressing this problem, because it requires experiment run-times which are too long to be practical for this project. As a result, the reward scheme used also applies a constant penalty for the robot coming into contact with the limits of the work envelope. This penalty helps to assure a Q-function which does not trap the agent early in an experiment.

This complication is at odds with the desire for a simple controller which does not require complex understanding of the interactions with and within the control plant. However, it is tolerated because the phenomenon being addressed is due to the manner in which the simulator is implemented, and thus would not be present in a real control situation.

#### 4.3.2.4 Selection of Action

The Q-learning algorithm used here only operates with discrete actions. Specifically, the selected **Policy**, the **EpsilonGreedyPolicy**, takes an **ActionSet** object which defines the possible actions from which the agent may choose. Hence, the **Action** from the **EpsilonGreedyPolicy** is output directly from the controller to the simulator stages.

#### 4.3.2.5 Tunable Parameters

The learning controller has a number of tunable parameters which can be used to optimize or modify the behaviour of the controller.

The **Policy** component has a single parameter:  $\epsilon$ , the ‘greediness’, or probability of performing an exploratory (rather than exploitive) action. The **Learner** component has three parameters:  $\alpha$  which specifies the learning rate (the proportion by which updates to the value of the Q-function for a specific state/action pair change the existing value of the Q-function);  $\gamma$  which gives the discount factor (the factor by which future rewards are discounted compared to immediate rewards); and  $\lambda$  (the rate at which eligibility traces decay). Additional information on these parameters is available in §2.2.2 and the accompanying references. Further, the parameters  $\sigma_\theta$  and  $\sigma_{\dot{\theta}}$  determine the ‘resolution’ of the ARBFN estimation of the Q-function. Finally, the reward scheme’s velocity gain  $K_{\dot{\theta}}$  must be selected. The reward scheme’s displacement gain  $K_\theta$ , is not an independent variable, because increasing it simply scales up all the reward values, and the *absolute* value of rewards is not important.

Thus, in total there are seven tunable parameters in this controller design; substantially more than for the traditional PID controller (see §4.3.1.2). This may appear at odds with the motivation behind using machine learning for a control system (see §1.1.3), because more parameters would suggest more design work. However, whilst the tunable parameters for the PID design directly determine the performance of the controller, this is not necessarily the case for the machine learning design. In the machine learning case, some of the tunable parameters may only affect the ‘transient’ performance of the controller whilst it is learning the environment, and other parameters may not adversely affect performance in a significant manner. Testing is required to determine whether this is the case.

## 4.4 Integration of Components

### 4.4.1 Modifications Required

The two libraries identified in §4.1 substantially reduce the software development required: the Teachingbox library can implement the machine learning algorithms of interest, whilst the Simbad library facilitates the creation of a simulator environment in which to test a control system using these algorithms. However, there remained work to be done in order to support the proposed simulation regime.

#### 4.4.1.1 Episodic Simulations

The Teachingbox library is generally intended for use with episodic experiments: an environment/agent pair is simulated over multiple time-steps, until the environment reaches some ‘goal state’. At this point, the ‘episode’ is complete, the environment is reset to a new initial state for the subsequent episode. As a rule, many episodes are required for learning to occur, so as to allow the agent to see a sufficiently large proportion of the state space for the environment.

The Simbad library was not originally intended for these types of episodic simulations; the simulator front-end provides functionality to run a single continuous simulation. To address this, an **EpisodicSimulator** class was created, which extends the original **Simulator** class from the Simbad library. The **EpisodicSimulator** class provides a mechanism to reset the simulator when the end of an episode is reached. The class also implements a counter which tracks the number of successful episodes occurring in each trial, as this is one of the primary metrics for evaluating controller performance.

#### 4.4.1.2 Triggers to Control Simulator

As well as restarting the simulator after each episode, a mechanism was also required to stop the simulation once the desired number of steps had been simulated. In order to implement both these functions, a system of ‘event triggers’ was created. Objects matching the interface **EpisodicSimulatorEventTrigger** could be added to the **EpisodicSimulator** instance through an **addEventTrigger()** method; added triggers were stored in a set within the simulator class. At each simulator time step, the list of associated triggers was evaluated; objects implementing the trigger interface were able to either stop or reset the simulator as required.

The **QuadSimbadAgent** class, which provides the implementation of the quadrotor agent within the simulator, was modified to implement the **EpisodicSimulatorEventTrigger** interface. When the control system connected to the simulated agent determined the agent had correctly achieved the desired goal state, the agent triggered the appropriate signal to command the **EpisodicSimulator** to end the current episode. To prevent ending an episode spuriously, the software was configured to require the agent to remain at the goal state for a minimum of five hundred steps prior to signalling the end of an episode. This ensures the control system must actually stabilize the quadrotor in the desired position instead of happening across the goal state by chance whilst throwing the vehicle around.

Similarly, the **QuadSimbadAgent** class was also configured to signal the simulator to stop once a sufficient number of simulator steps had been completed. This functionality was implemented

in the agent rather than in the simulator itself, minimizing the amount of extension required of existing classes from the Simbad library.

#### 4.4.1.3 Separation of Robot & Controller

The Simbad library is designed to simulate the behaviour of one or more ‘robots’. Generally, these robots would be implemented through a class which determines both the simulated physical properties of the robot, its actuators and sensors; and also the logical behaviour of the robot’s controller. In this project, however, it was desirable to separate the functionality required to simulate the robot and that which implements the control systems being tested.

This was achieved through the creation of a **SimbadRobotController** interface, which allows a control system to be connected to the **QuadSimbadAgent** class which implements the simulation of the quadrotor, without requiring any modification to the agent class. The interface allows a controller to interact with a simulated robot through a single ‘performBehavior’ method, to which the simulator provides details of the current state of the robot, and from which the controller returns the desired action to be taken.

A **SimbadLearningRobotController** class was created to implement the interface, allowing connection from the simulator to an ‘Agent’ class as implemented by the Teachingbox library. The **SimbadLearningRobotController** performs much of the same functionality as the native **Experiment** class in the Teachingbox library, but allows the use of a more sophisticated simulator environment. By creating an instance of the **SimbadRobotController** class with a specific Teachingbox agent, a robot in the Simbad simulator may be controlled using algorithms from the Teachingbox library.

This arrangement permits the use of multiple distinct control systems to independently control differing aspects of a single robot. In this project, three **SimbadLearningRobotController** instances were used to independently control the three rotational axes of the simulated quadrotor.

#### 4.4.1.4 Packaging of States & Actions

The Teachingbox library provides specific classes to encapsulate the concepts of ‘states’ and ‘actions’, since these are important in the application of machine learning algorithms. The Simbad library, however, uses instead the vector maths classes available in the **VecMath** package: for instance, the class **Vector3d** encodes translational state information. This is appropriate because, whilst the Simbad simulator is only intended for use with 2D or 3D simulated environments, the Teachingbox library is more generic, and may be used with environments which require more terms to describe their state.

Accordingly, the **QuadSimbadAgent** class converts to and from the Teachingbox specific **State** and **Action** objects, and appropriately sized **Vector** objects as required. This allows for splitting out the elements describing the state of each axis of the simulated robot for the independent control systems associated with each axis.

#### 4.4.2 Typical Call Chain

The `QuadSimbadExperiment` class provides a `main()` function which can be executed to initiate the simulator. This sets up logging functionality. The function then creates an instance of `QuadSimbadEnvironment`, which defines the 3D ‘environment’ in which the simulation will be performed; an instance of `QuadSimbadAgent`, which defines the ‘robot’ to be simulated; and finally a new instance of `EpisodicSimulator`, the implementation of the simulator itself.

The `EpisodicSimulator` creates a Swing graphical user interface, which features a rendered view of the simulator environment, and a number of controls through which to administer the simulator.

Once simulation is started, a timed loop performs simulator ‘steps’ repeatedly until either stopped by the user, or an internal trigger occurs which stops the simulation. At each simulation step, the simulator class calls the `performBehavior()` method of the agent for each robot present in the simulation. This allows the agent to perform control behaviour as required. For each agent, the selected `KinematicsModel` class (in this case, `QuadSimbadKinematicsModel`) is used to simulate the dynamics of the robot. Then the Agent’s `updatePosition()` method actually moves the model of the robot within the 3D simulation environment.

In `QuadSimbadAgent`, the `performBehaviour()` method calls the child `performBehavior()` methods for the `SimbadRobotController` objects assigned to control each of the quadrotor’s individual axes. The results from each controller are mixed according to the motor mixing algorithm, then sent to the kinematics implementation so that the dynamics of the quadrotor will be adjusted in the next simulator time-step.

In the `SimbadLearningRobotController` implementation, the `performBehaviour()` method calls the `nextStep()` method of an attached Teachingbox `Agent` object; it also provides a notification to any attached ‘listener’ objects, and calculates the reward associated with the current time-step, to be passed to the learning agent.

The selection and configuration of a TeachingBox `Agent` to use for each experiment is performed during the initialisation of the `QuadSimbadEnvironment` object. The constructor of the class `QuadSimbadEnvironment` creates an instance of an `Agent` for each of the individual controller axes, using whatever control algorithm is required for the particular experiment. Each agent encapsulates a `Policy` object, which maps observed `States` onto controller `Actions`. Additionally, when using a machine-learning algorithm, the agent also encapsulates a `Learner` object, which observes the environment and received rewards in order to develop improved policy.

A simplified UML diagram of the manner in which the software is constructed is shown as Figure 4.3.

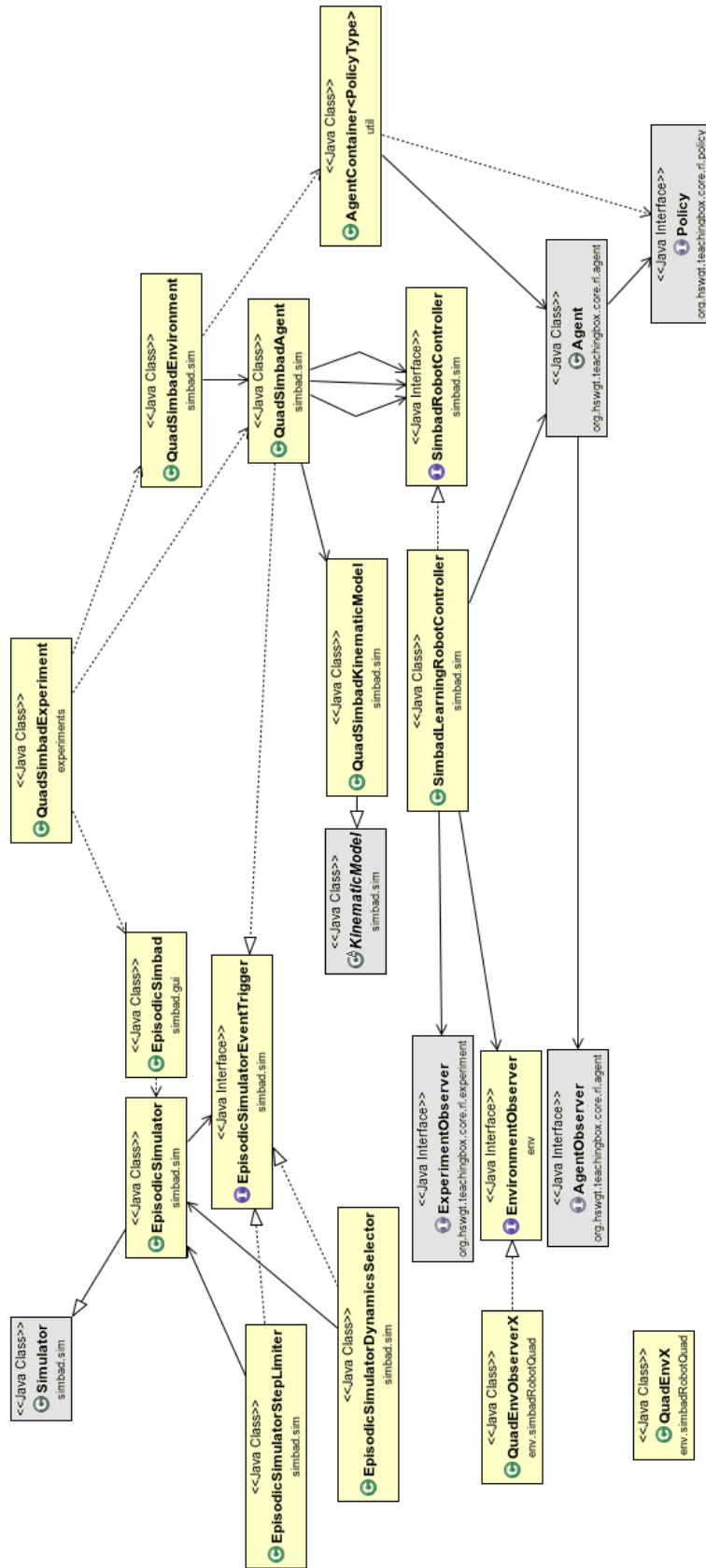


Figure 4.3: UML Diagram of Simulator Components: Grey components belong to the Simbad and Teachingbox libraries; yellow components were created during this project.



## Chapter 5

# Performance of PID Controller

This chapter details a series of tests which will characterise the performance of a conventional PID control system. The collated results are then analysed.

### 5.1 Experimental Details

Experiments were performed using the PID control algorithm (as detailed in §4.3.1), in order to create a baseline against which the performance of the machine learning based algorithm could be evaluated. Broadly, the significant area of interest is how control performance varies as a function of the effort the developer exerts on tuning the controller. Comparison against the proposed reinforcement learning based controller will allow assessment of whether that algorithm might be able to realise some advantages over the conventional PID design.

#### 5.1.1 Test Procedure

Each experiment consists of a series of three trials. Whilst three trials alone are not very statistically reliable, the behaviour of PID controllers is deterministic and well understood. Thus three trials is understood to be sufficient for indicative purposes.

In each trial, five million time-steps are simulated (corresponding to 55.5 hours of simulated operation, at 25 frames per second). Each trial is broken down into a number of *episodes* (see §4.4.1.1), to ensure that as much as possible of the state space for the environment is simulated: at the start of each episode, the quadrotor begins in a (uniformly) randomly selected orientation within the working volume. The controller attempts to return the quadrotor to the *nominal* (flat) position. Once the quadrotor has been within some small threshold of the nominal position for a continuous period of 500 time steps (20 seconds in simulator time), the episode is declared complete and a new episode begins.

During each trial, the simulator state at each time step is logged to a text file (approximately 700MB of storage per trial). This allows the entire simulation to be recreated later if required. The log file is then parsed to extract key performance indicators for analysis.

Experiments are performed with a range of tunable parameter values, so as to baseline data for the under-damped, critically-damped and over-damped controller cases.

### 5.1.2 Performance Metrics

The primary performance metric against which each trial is analysed is the mean length (in simulator steps) of each episode; the length of each episode is given by the number of steps taken before the robot remains close to the nominal position for 500 consecutive steps, so episode length directly correlates with ‘settling time’ (as used to describe the performance of a conventional second order control system). Lower values are therefore better (with 500 being the smallest possible value).

Because the controller uses a conventional PID algorithm which has no ‘learning’ behaviour, it is expected that the response of the controller remains constant throughout each trial. Accordingly, the behaviour of the controller at any point in the trial can be considered indicative.

Because the initial conditions for each episode are uniformly distributed, episode length varies across episodes; hence the median episode length is used as an indication of overall performance during an episode. However, since in many control applications even a single poor response may be considered unacceptable, the standard deviation and other statistical descriptors of episode length should also be considered.

Whilst more complicated analysis is possible, this single metric of episode length is considered sufficient to compare the performance between different trials; it strongly captures stability (of greatest concern in terms of evaluating the viability of a control algorithm) whilst also capturing the speed of response to a step input.

Additionally, a number of qualitative factors are considered, including the shape of the time-domain step response of the controller during each episode.

## 5.2 Results

### 5.2.1 A First Attempt

A set of initial gains for the PID controller were chosen intuitively:  $K_p = 0.5$ ,  $K_i = 0.0$ ,  $K_d = 5.0$ . A set of three simulations were performed with the controller configured using these gains. The resulting episode lengths for each trial are plotted in Figure 5.1. Note that this figure, and all similar figures in this chapter, are plotted against a linear scale on the vertical axis.

The plot shows that, as expected, the behaviour of the controller remains constant throughout each trial (as the PID controller has no learning functionality). Also, as expected, the episode length varies from episode to episode because the initial conditions of each episode are uniformly distributed (though it is noted that this does not translate directly into a uniform distribution of episode lengths). To better display the performance characteristics from each trial, the results are compiled into the box plots shown in Figure 5.2.

The first three columns of Figure 5.2 clearly show that the performance of all three trials is essentially identical. In light of this, and because the behaviour of the controller remains constant throughout each trial, the results can be aggregated into a single result for the experiment by simply concatenating the three trials together; this is shown in the fourth column. The individual and aggregate performance statistics are also shown in Table 5.1. Henceforth, this aggregate approach will be used to analyse the results of each experiment.

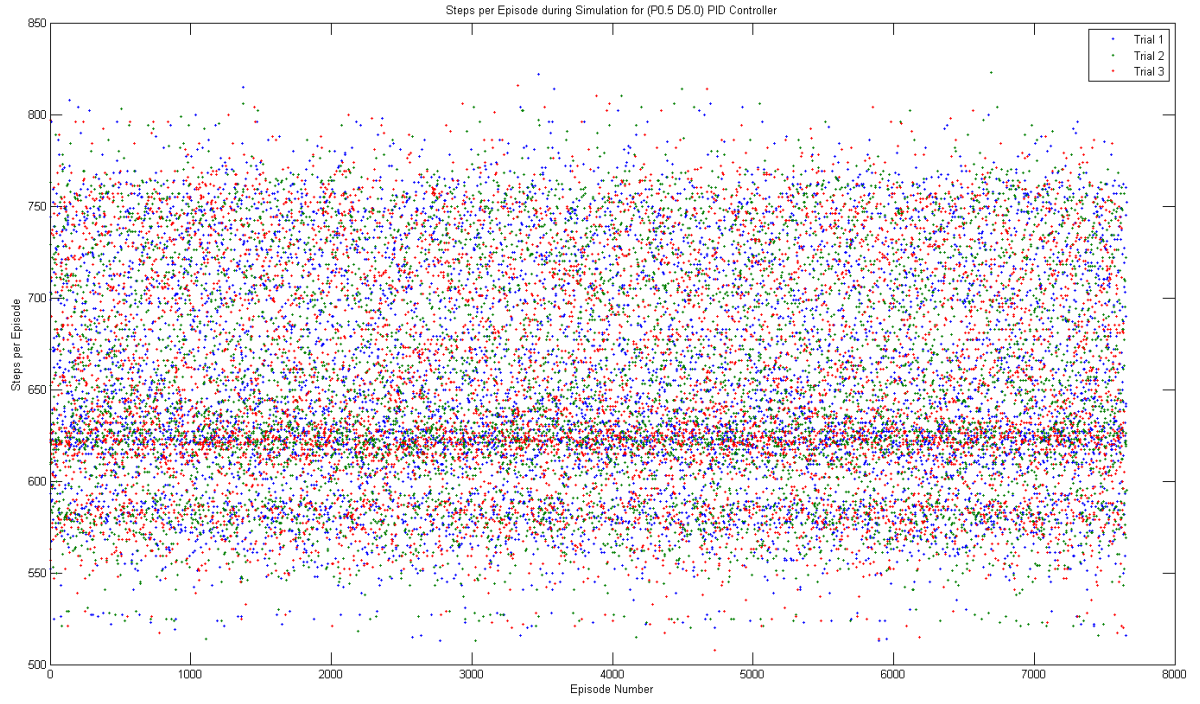


Figure 5.1: Individual episode lengths for PID controller ( $K_p = 0.5$ ,  $K_d = 5.0$ ).

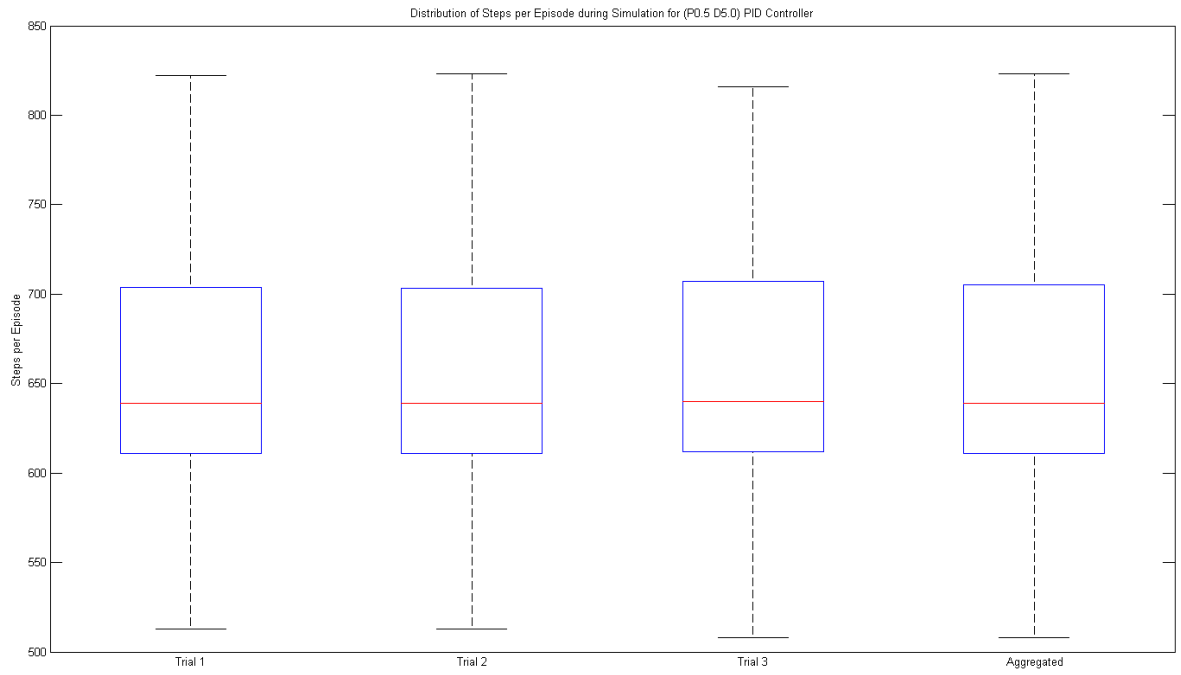


Figure 5.2: Compiled episode lengths for PID controller ( $K_p = 0.5$ ,  $K_d = 5.0$ ).

Finally, the actual time-domain response of the controller should be considered. A plot of sample responses for each of the three controlled axes are shown in Figure 5.3: a section of log data for each axis was selected randomly, (hence, the responses shown for each axis in the plot are unrelated to the responses of the other axes). The division of the response into individual episodes is clear; the simulator selects a random initial position which appears as a step input, the controller corrects the error, and once the error has remained sufficiently small for at least 500 steps, a new episode begins. As expected, the response displays characteristics of a second order system. The chosen controller gains result in the system being heavily underdamped.

## 5.2.2 Other Controller Gains

Because the software is configured as a ‘bang-bang’ controller (see §4.3.1.1), there is zero steady-state error from a step input. Accordingly, no integral action is necessary; hence  $K_i$  is kept as zero throughout the experiments. Additionally, the absolute value of  $K_p$  is unimportant, only the relative difference between  $K_p$  and  $K_d$ . This leaves  $K_d$  as the only parameter which requires adjusting in this example.

Because the first experiment resulted in a clearly underdamped response, the controller gains are adjusted to incorporate additional damping. An experiment is conducted for gains of  $K_d = 15.0$  and then  $K_d = 10.0$ . As previously, the three trials in each experiment are aggregated; results from each experiment are shown in Figure 5.4, as well as being listed in Table 5.1.

When using gains of  $K_p = 0.5$  and  $K_d = 15.0$ , there is a slight improvement in median episode length, and significant improvement in the top-end distribution of episode lengths. A sample of the time-domain response of the controller using this set of gains is shown in Figure 5.5. In this plot, it is clear that the new controller gains have resulted in an overdamped system.

When using gains of  $K_p = 0.5$  and  $K_d = 10.0$ , there is further notable improvement in median episode length. A sample of the time-domain response of the controller using this set of gains is shown in Figure 5.6. In this plot, the controller response appears to be approximately critically damped (there being slight overdamping in some episodes, and slight underdamping in others). Accordingly, the performance of the PID controller using this set of gains will be used as a baseline for assessment of performance during further experiments.

Gain $K_d$ Trial	Experiment											
	5.0				15.0				10.0			
	1	2	3		1	2	3		1	2	3	
Total Episodes	7656	7656	7640	22952	7956	7951	7957	23864	7960	7960	7960	23880
Mean	653	653	654	653	628	629	628	629	599	599	599	599
Median	639	639	640	639	630	631	630	630	589	589	589	589
Std Dev	60.7	60.8	61.0	60.8	22.9	22.7	23.2	22.9	30.6	30.7	30.4	30.6
Lower Qtl	611	611	612	611	610	610	610	610	575	575	575	575
Upper Qtl	704	703	707	705	649	649	649	649	624	624	622	624
Minimum	513	513	508	508	506	526	502	502	517	513	522	513
Maximum	822	823	816	823	673	671	675	675	674	674	672	674

Table 5.1: Episode length statistics for experiments varying PID controller gains (aggregate results for each experiment are shown shaded).

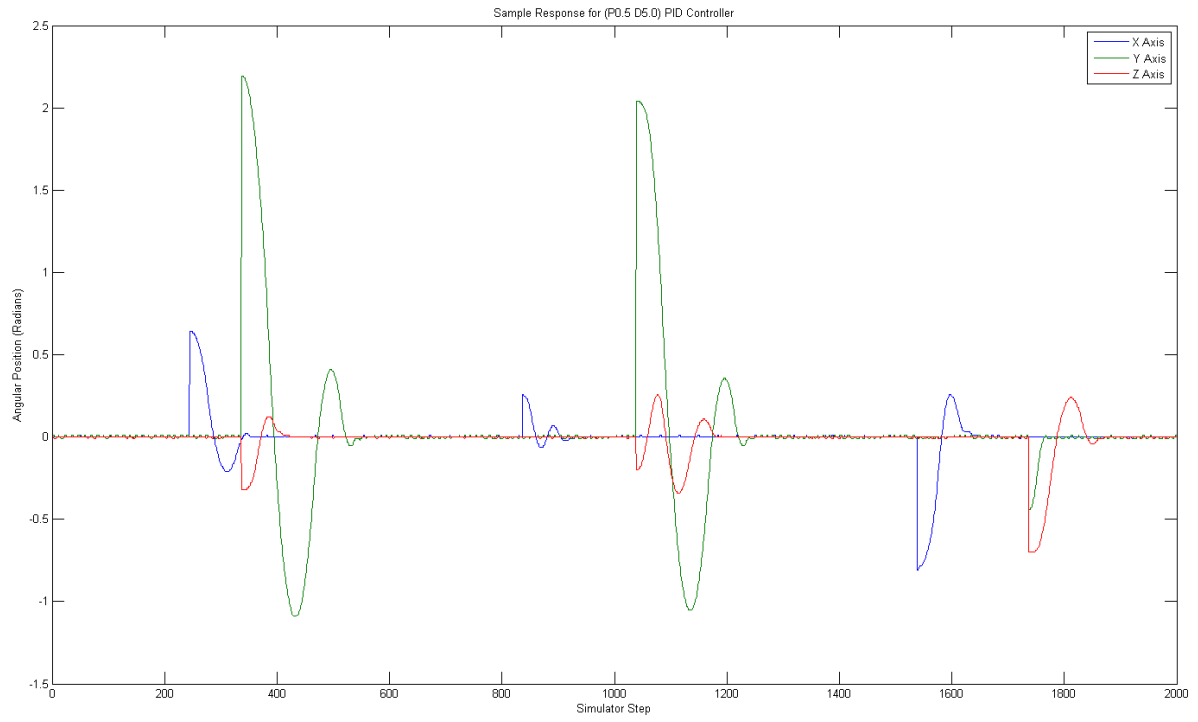


Figure 5.3: Sample time-domain response for PID controller ( $K_p = 0.5$ ,  $K_d = 5.0$ ).

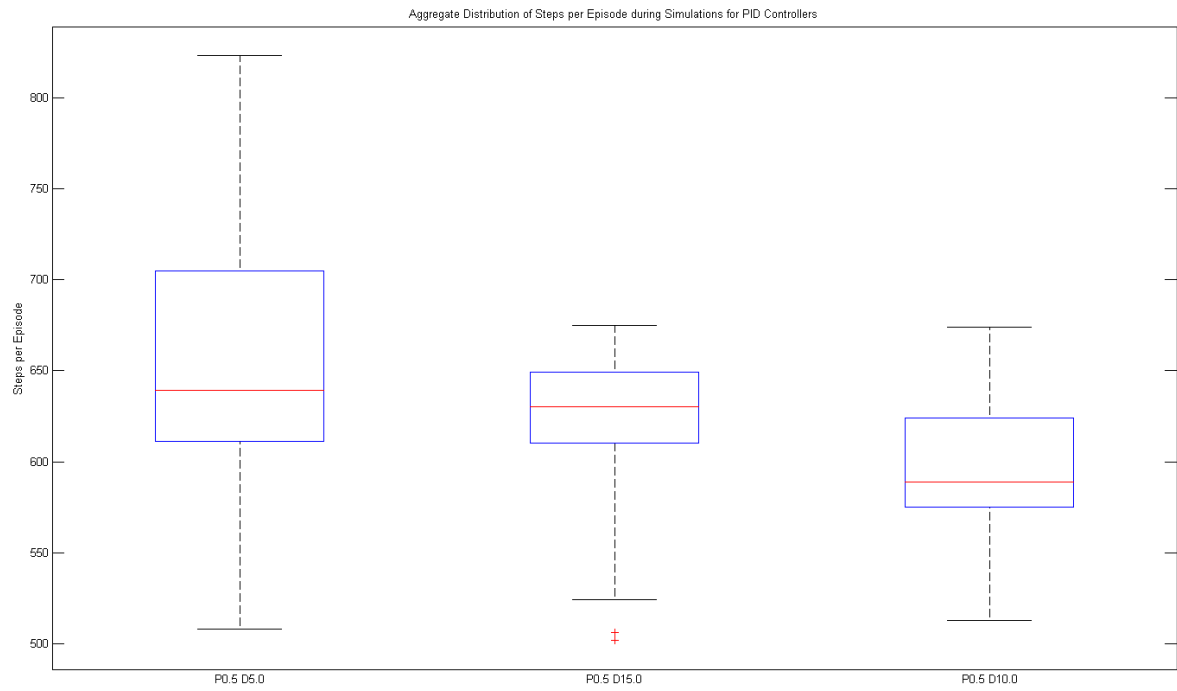


Figure 5.4: Compiled episode lengths for PID controller with varying gains.

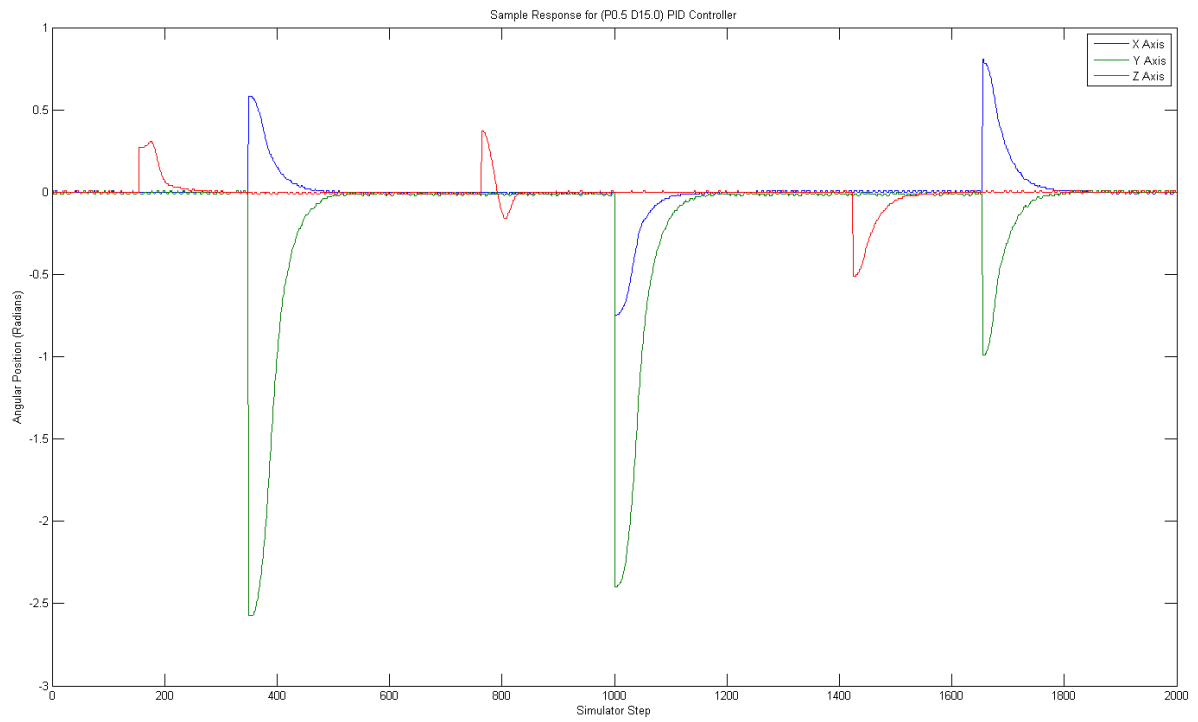


Figure 5.5: Sample time-domain response for PID controller ( $K_p = 0.5$ ,  $K_d = 15.0$ ).

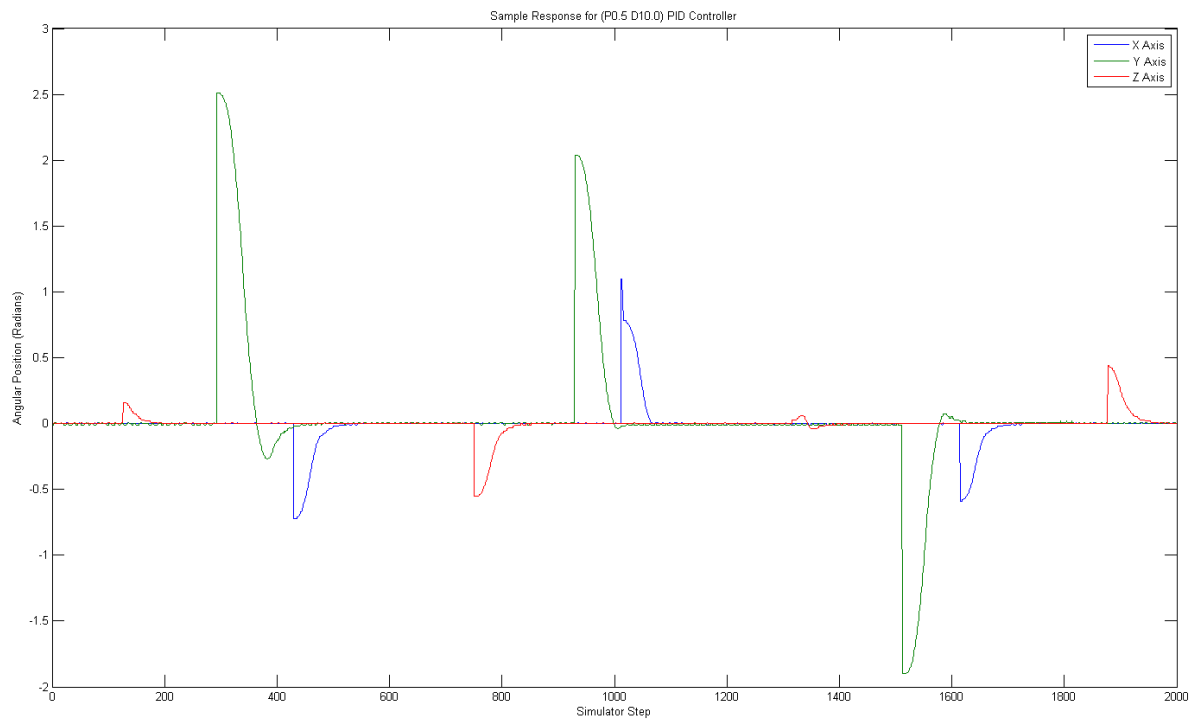


Figure 5.6: Sample time-domain response for PID controller ( $K_p = 0.5$ ,  $K_d = 10.0$ ).

### 5.2.3 Effects of Disturbance

Because any practical control system must be able to handle disturbances applied to the plant, an additional experiment was performed to characterise the performance of the PID controller in the presence of disturbances.

Disturbance was simulated through the addition of a randomly distributed moment being applied to the vehicle at each time step. The disturbance component was independently normally distributed in each axis, with a mean of zero, and a standard deviation of  $2.5Nm^{-1}$  (comparable with the maximum control moment available to the controller, which is around  $5.5Nm^{-1}$ ). This represents an environment with strong disturbances, such as might be encountered during flight in turbulent conditions. The ability of the controller to remain stable in the presence of such disturbances is a necessary characteristic for any prospective flight control system.

It is worth recognising that a step-wise normally distributed disturbance, as used here, is unlikely to be a very accurate model of the disturbances encountered in reality for a flight controller. More likely, any disturbance will have some systemic characteristics in addition to random noise. However, for the purposes of determining how the stability and performance of the controller is affected in the presence of disturbance, this simplified model should be sufficient.

A set of three trials were simulated, using the gain set selected as a baseline in the previous experiment ( $K_p = 0.5$  and  $K_d = 10.0$ ). In these trials, normally distributed disturbances with standard deviation of 2.5 were present. As previously, the three trials in the experiment were aggregated together; the results from the experiment are shown in Figure 5.7, and the results are also listed in Table 5.2. For comparative purposes, the box plot for the undisturbed case (as detailed in §5.2.2) is also shown in Figure 5.7.

In the presence of disturbances, there is significant degradation of the controller performance in terms of episode length. The main factor in this reduction in performance is most likely that the derivative component of the controller is adversely affected by increased noise: no filtering is performed on the derivative signal which is calculated by the controller, so the random impulses injected by the disturbances in this experiment result in a very noisy estimation of error derivative. In a practical controller, this noise could be filtered reasonably easily, but the situation would need to be identified first.

However, consideration of a sample of the time-domain response of the controller, as shown in Figure 5.8, shows that the control response settles to be fairly close to the nominal position in around 100 simulator steps, which is not substantially different to the response seen in the undisturbed case. However, the presence of disturbance means that it can take substantially

Trial	Without Disturbances				With Disturbances			
	1	2	3		1	2	3	
Total Episodes	7960	7960	7960	23880	3003	3073	3039	9115
Mean	599	599	599	599	1665	1627	1645	1646
Median	589	589	589	589	1319	1241	1284	1280
Std Dev	30.6	30.7	30.4	30.6	1173.2	1165.6	1161.6	1166.9
Lower Qtl	575	575	575	575	821	783	788	798
Upper Qtl	624	624	622	624	2107	2071	2093	2090
Minimum	517	513	522	502	534	536	533	533
Maximum	674	674	672	674	10871	10514	8916	10871

Table 5.2: Episode length statistics for experiments with and without disturbances (aggregate results for each experiment are shown shaded).

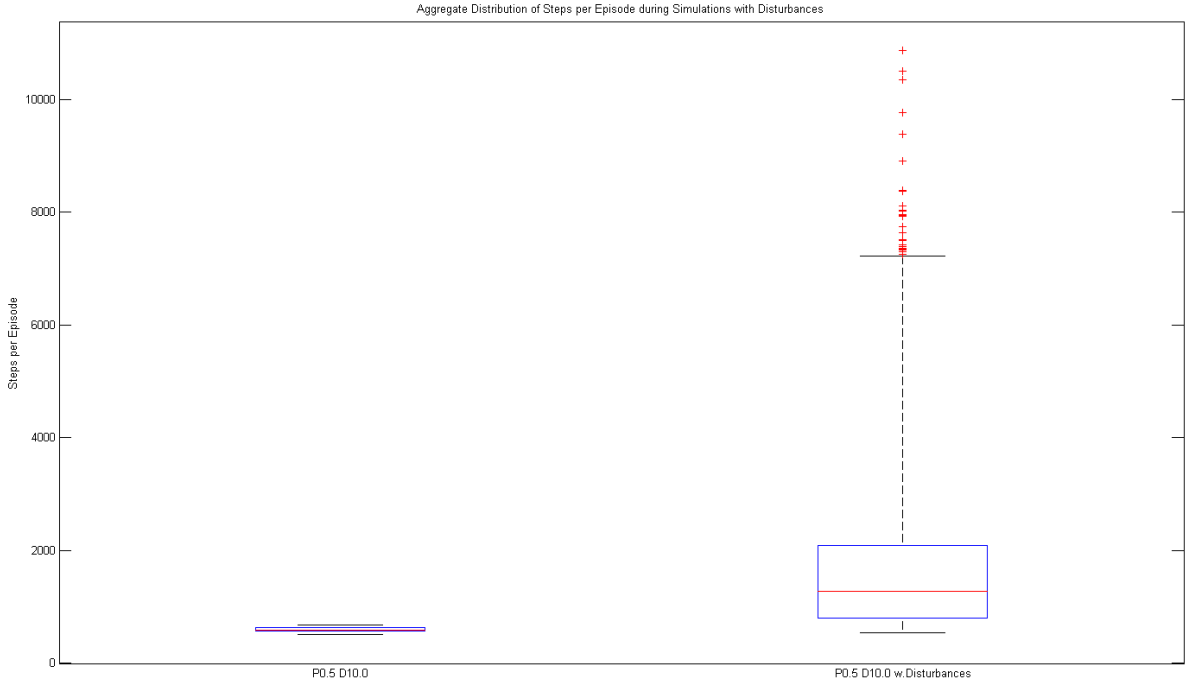


Figure 5.7: Compiled episode lengths for PID controller in the presence of disturbances.

longer for the controller to achieve the required number of sequential steps without the error growing larger than the threshold value, even if momentarily.

#### 5.2.4 Handling Changes in Dynamics

As discussed in §1.1, dealing with variations in plant dynamics is a significant issue in control system design; it is hoped that use of RL based algorithms could be beneficial in this regard. To assess any such benefit, a baseline is required for comparison. A further set of experiments was thus performed to characterise the performance of the PID controller in the case where the plant dynamics are modified at runtime.

In these experiments, the first half of each trial (i.e. the first two and a half million steps) was conducted using the same simulator configuration as used previously. At the mid-point of each trial, the plant dynamics were adjusted by increasing the mass of the vehicle by a factor of three (from  $m = 10$  to  $m = 30$ ). The second half of the trial was then performed using the original controller instance, operating on the newly adjusted dynamics.

Two experiments of three trials each were performed, to characterise control performance both with and without the presence of disturbances.

##### 5.2.4.1 Without Disturbances

In this experiment, the PID controller used the gain set previously selected as a baseline. The resulting episode lengths for each trial are shown plotted in Figure 5.9. The plot shows that,



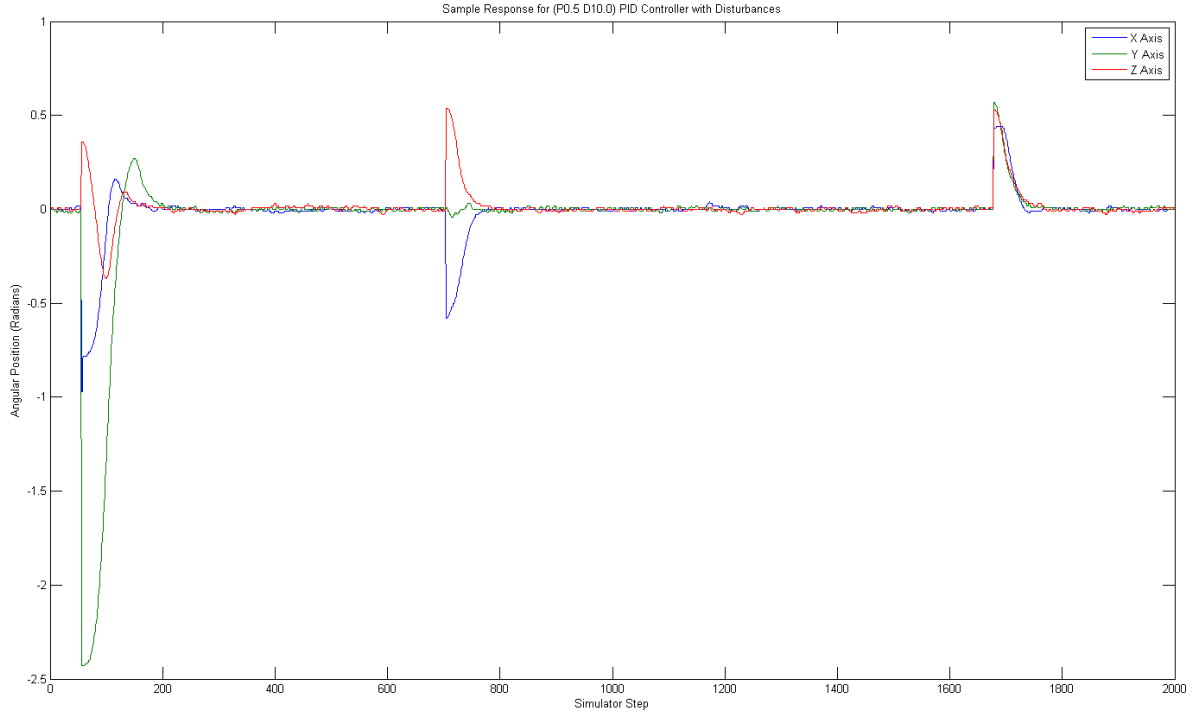


Figure 5.8: Sample time-domain response for PID controller with disturbances ( $K_p = 0.5$ ,  $K_d = 10.0$ ).

as expected, the behaviour of the controller remains constant throughout each trial (as the PID controller has no learning functionality), up until the point at which the plant dynamics are modified. When the plant dynamics are modified, there is a step change in the performance of the PID controller, but following this change, the performance remains constant throughout the remainder of the trial. Also as expected, the episode length varies from episode to episode because the initial conditions of each episode are uniformly distributed (though note that this does not translate directly into a uniform distribution of episode lengths).

To better display the performance characteristics from each trial, the results for each trial are compiled into the box plots shown in Figure 5.10. To produce these aggregate results, data from each trial was divided into ‘before’ and ‘after’ regions, which were then concatenated together as in previous experiments. The aggregate performance statistics are also shown in Table 5.3.

Trial	Without Disturbances		With Disturbances	
	Before ( $m = 10$ )	After ( $m = 30$ )	Before ( $m = 10$ )	After ( $m = 30$ )
Total Episodes	12578	10625	8219	5649
Mean	596	705	914	1324
Median	586	682	729	1075
Std Dev	29.6	81.3	441.5	771.4
Lower Qtl	573	652	601	785
Upper Qtl	619	774	1074	1609
Minimum	513	521	528	534
Maximum	676	897	3995	7405

Table 5.3: Episode length statistics for experiments modifying plant dynamics (aggregate results).

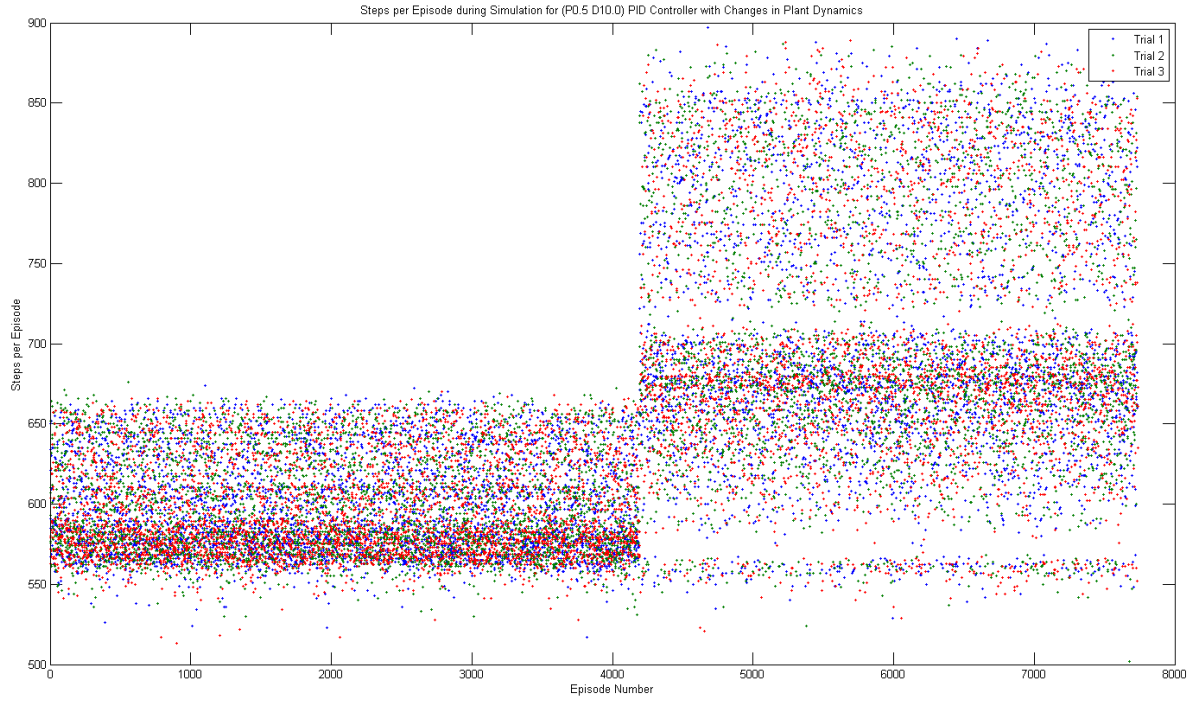


Figure 5.9: Individual episode lengths for PID controller ( $K_p = 0.5$ ,  $K_d = 10.0$ ) with modified plant dynamics.

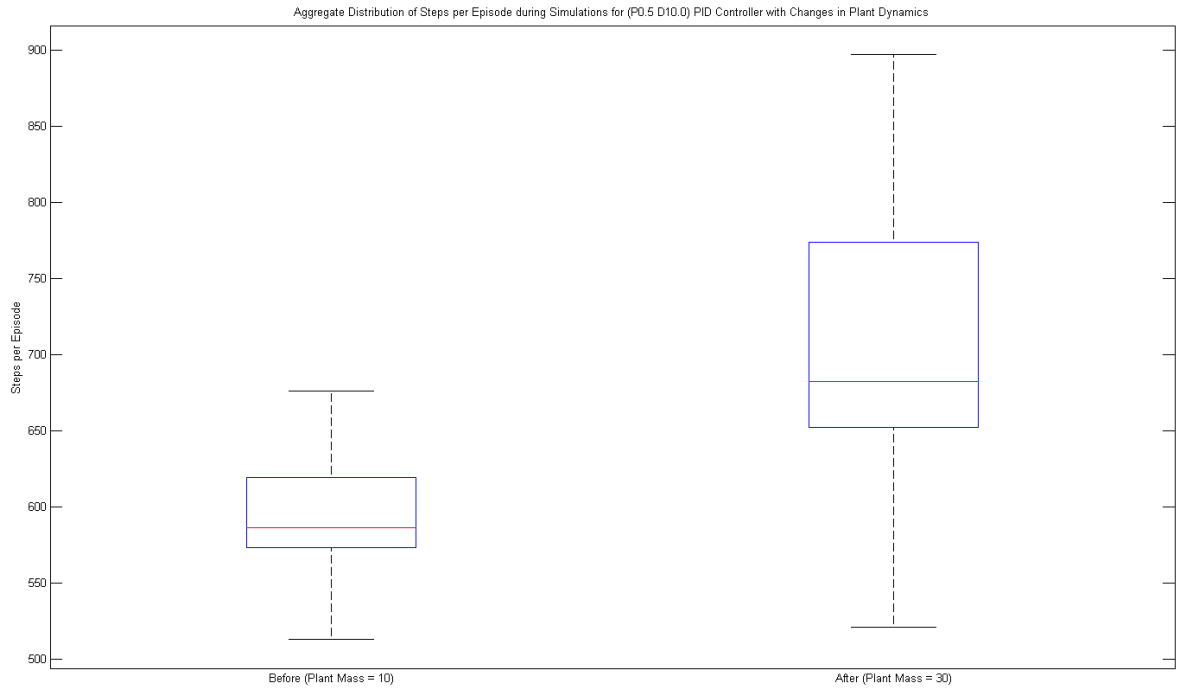


Figure 5.10: Compiled episode lengths for PID controller ( $K_p = 0.5$ ,  $K_d = 10.0$ ) with modified plant dynamics.

Finally, the actual time-domain response of the controller should be considered. A plot of sample responses for each of the three controlled axes, before and after the modification in plant dynamics, are shown in Figures 5.11 and 5.12, respectively: a section of log data for each axis was selected randomly from within each of the two dynamics regions. As seen previously, the response of the controller prior to the change in plant mass is roughly critically damped. The response of the controller following the change in plant mass is clearly under-damped, leading to the increased episode length.

#### 5.2.4.2 With Disturbances

In this experiment, the PID controller used the gain set previously selected as a baseline. As in the experiment detailed in §5.2.3, a normally distributed moment was applied to the vehicle to simulate an external disturbance. The resulting episode lengths for each trial are shown plotted in Figure 5.13. The plot shows that, as expected, the behaviour of the controller remains constant throughout each trial (as the PID controller has no learning functionality), up until the point at which the plant dynamics are modified. When the plant dynamics are modified, there is a step change in the performance of the PID controller, but following this change, the performance remains constant throughout the remainder of the trial. Also as expected, the episode length varies considerably from episode to episode due to the distribution of initial conditions for each episode and the additional disturbances.

To better display the performance characteristics from each trial, the results for each trial are compiled into the box plots shown in Figure 5.14. To produce these aggregate results, data from each trial was divided into ‘before’ and ‘after’ regions, which were then concatenated together as in previous experiments. The aggregate performance statistics are also shown in Table 5.3.

Finally, the actual time-domain response of the controller should be considered. A plot of sample responses for each of the three controlled axes, before and after the modification in plant dynamics are shown in Figures 5.15 and 5.16, respectively: a section of log data for each axis was selected randomly from within each of the two dynamics regions. As seen previously, the response of the controller prior to the change in plant mass is roughly critically damped. The response of the controller following the change in plant mass is clearly severely under-damped, leading to the increased episode length.

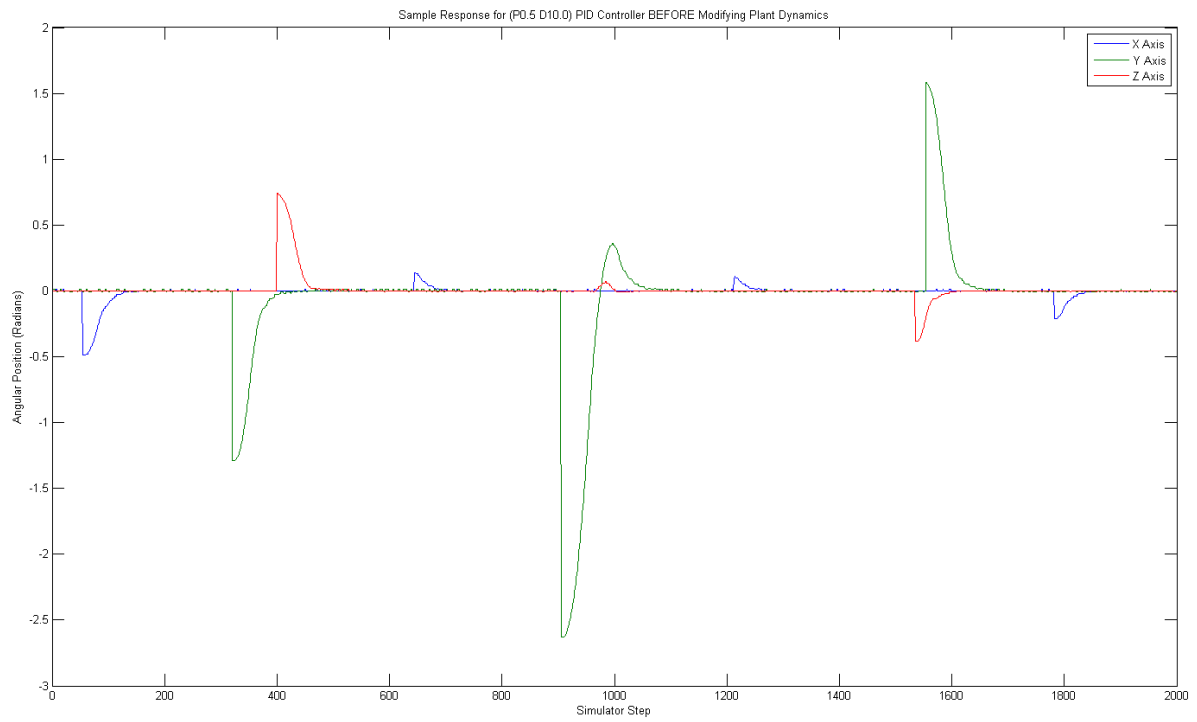


Figure 5.11: Sample time-domain response for PID controller ( $K_p = 0.5$ ,  $K_d = 10.0$ ) before modifying plant dynamics.

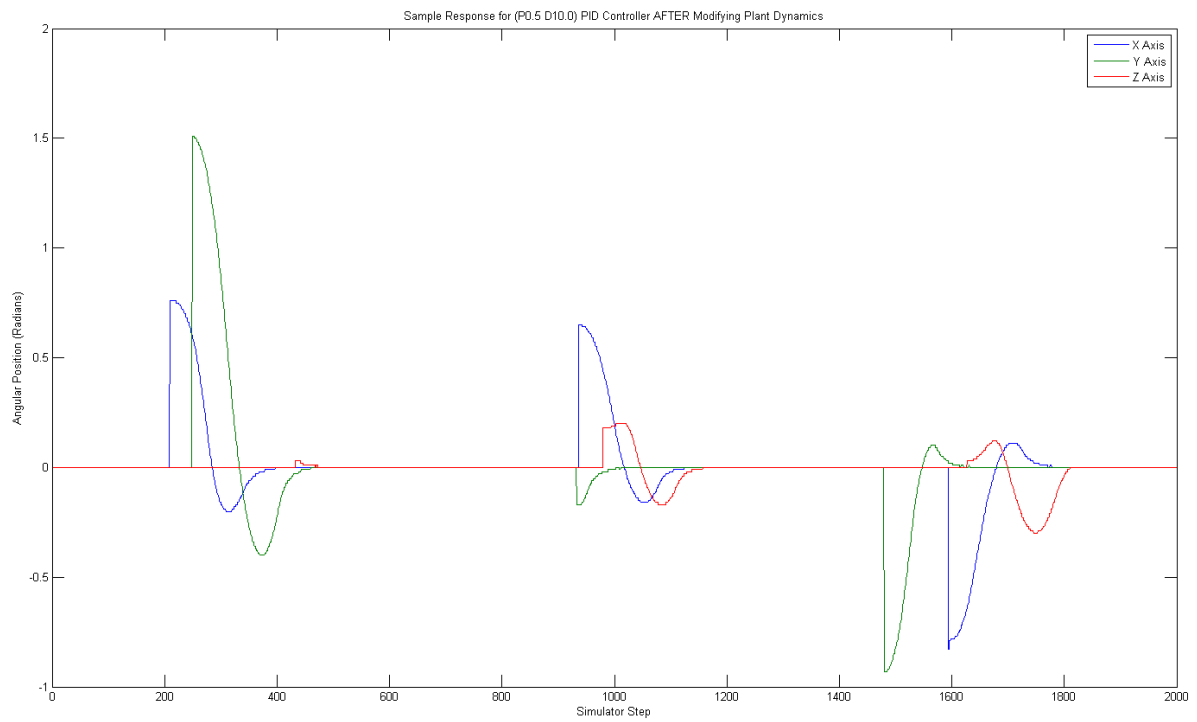


Figure 5.12: Sample time-domain response for PID controller ( $K_p = 0.5$ ,  $K_d = 10.0$ ) after modifying plant dynamics.

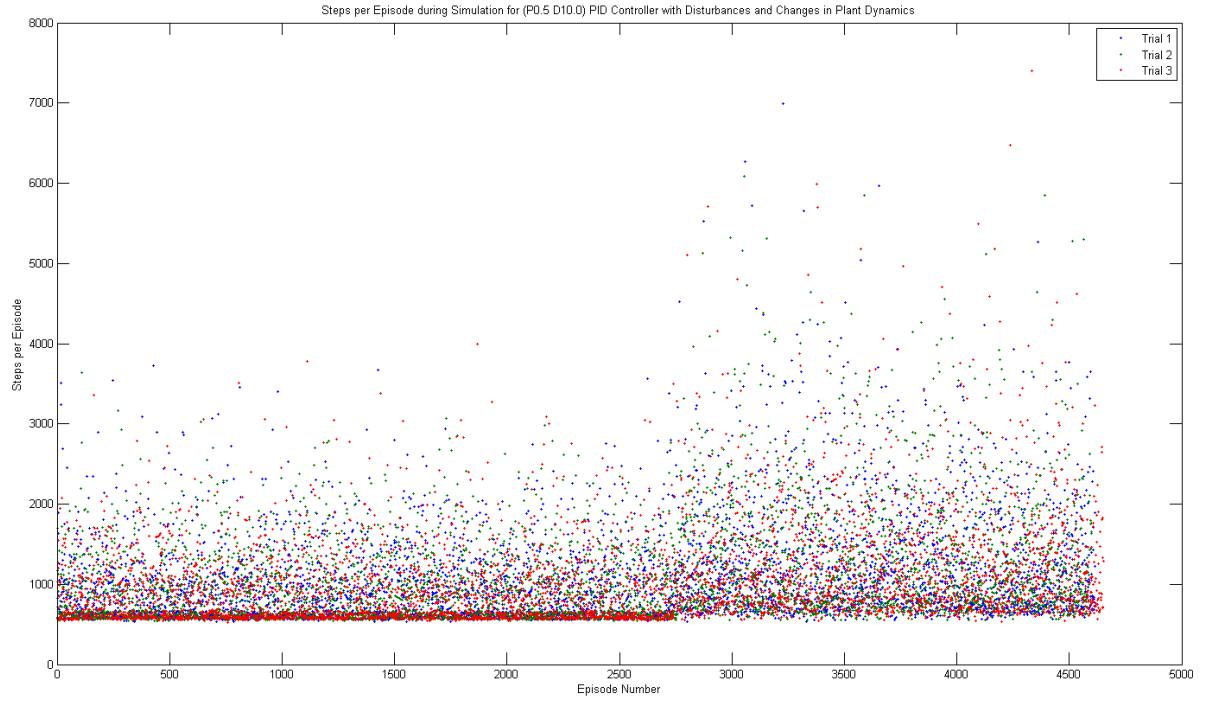


Figure 5.13: Individual episode lengths for PID controller ( $K_p = 0.5$ ,  $K_d = 10.0$ ) with modified plant dynamics and disturbances.

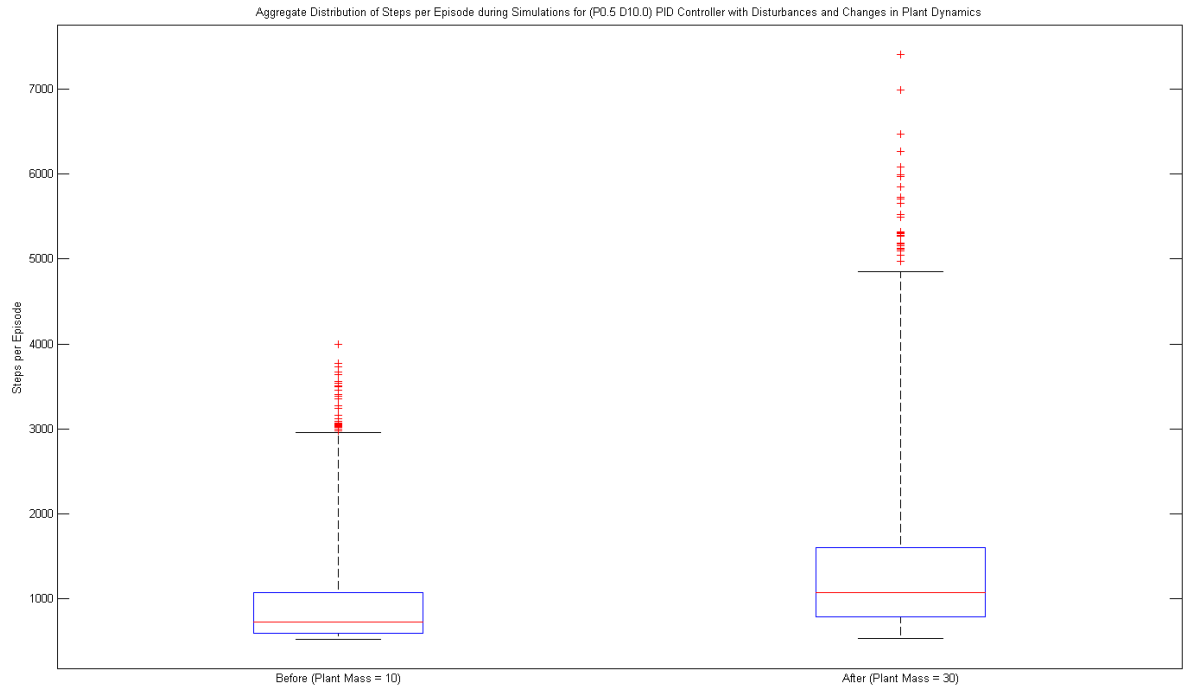


Figure 5.14: Compiled episode lengths for PID controller ( $K_p = 0.5$ ,  $K_d = 10.0$ ) with modified plant dynamics and disturbances.

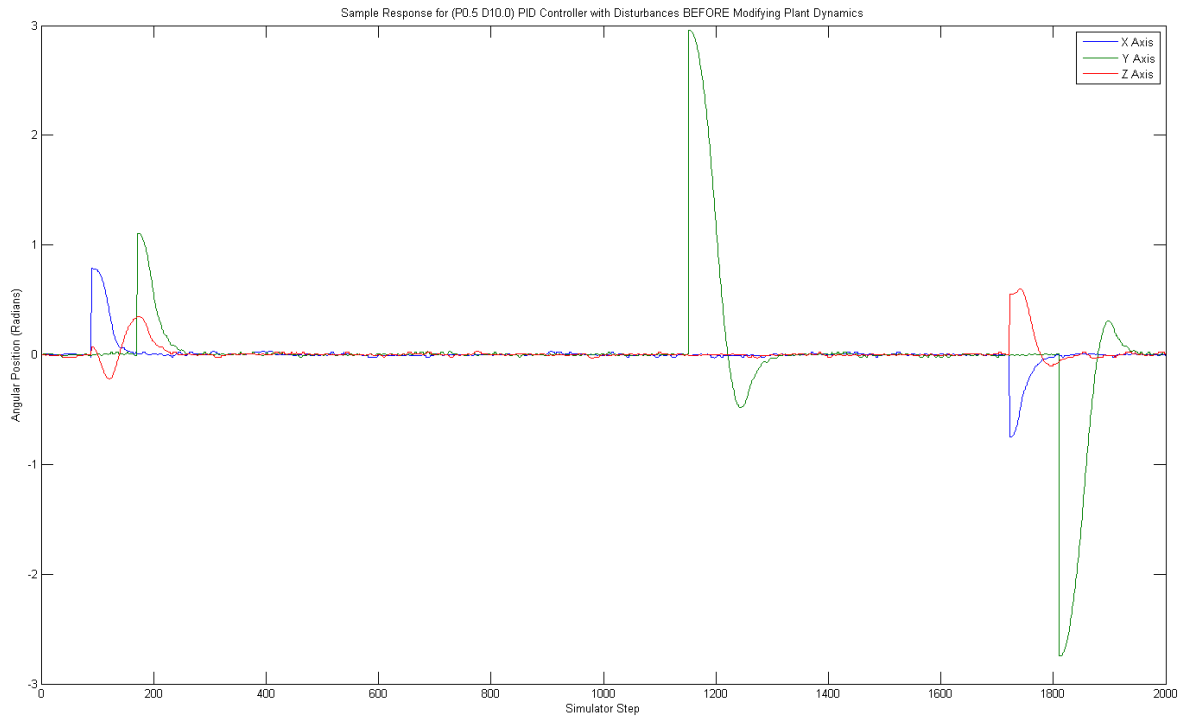


Figure 5.15: Sample time-domain response for PID controller ( $K_p = 0.5$ ,  $K_d = 10.0$ ) with disturbances before modifying plant dynamics.

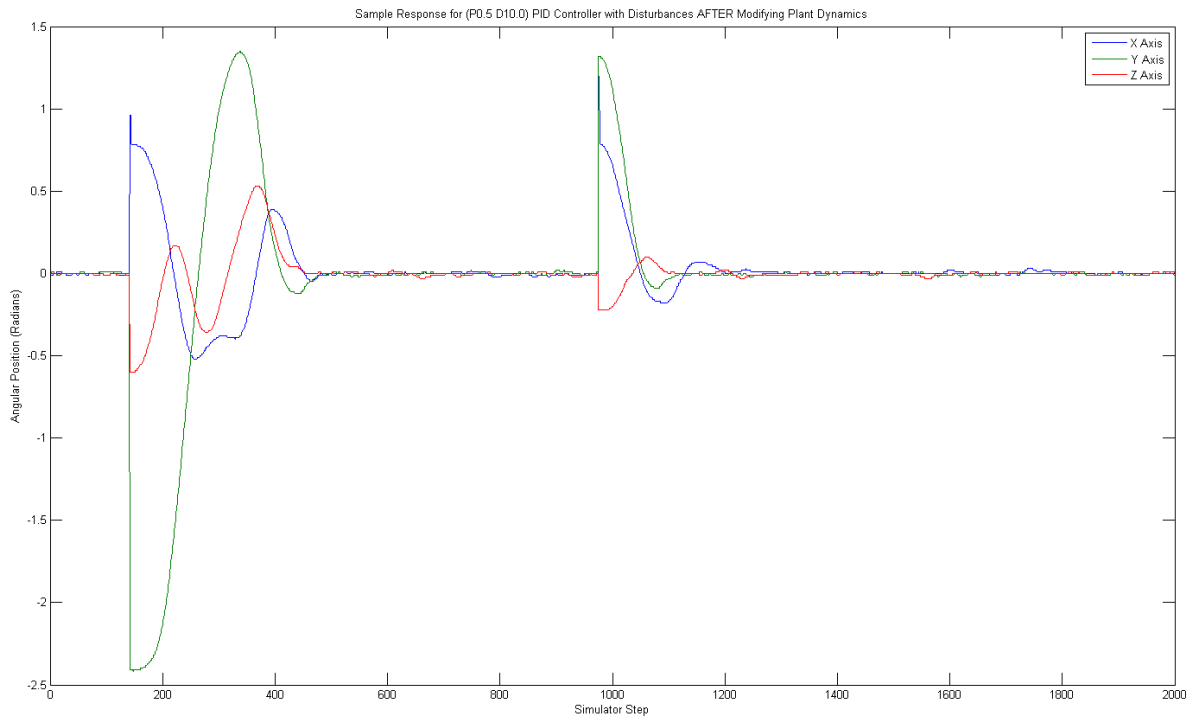


Figure 5.16: Sample time-domain response for PID controller ( $K_p = 0.5$ ,  $K_d = 10.0$ ) with disturbances after modifying plant dynamics.

## Chapter 6

# Performance of Q-Learning Based Controller

In this chapter, a series of tests which will characterise the performance of the prototype RL based control system are detailed. Tests are performed to both evaluate the performance of the controller, and characterise the affect of varying controller parameters on performance. The results of these tests are then collated and analysed.

### 6.1 Experimental Details

A series of experiments were performed using the Q-Learning based control algorithm (as detailed in §4.3.2), in order to evaluate how well the algorithm performs. Broadly, it is desired to see how the algorithm performs when using both a randomly chosen (or default) set of tunable parameters (see §4.3.2.5), and a set of optimal parameters. This will allow estimation of some bounds on the control performance which can be expected from a Q-Learning based control system, and of how much developer effort is required in terms of tuning, which in turn will allow assessment of whether the algorithm might be able to offer the benefits outlined in §1.1.3: improved control performance and reduced development time.

#### 6.1.1 Test Procedure

Experimental procedure for testing the Q-Learning based controller was the same as for the PID controller, as detailed in §5.1.1.

Each experiment consists of a series of three trials, with each trial containing five million time-steps. Each trial is broken down into a number of episodes: the quadrotor begins in a randomly selected orientation; the controller attempts to return the quadrotor to the flat position; once the quadrotor has been close to the nominal position for some continuous period, the episode is declared complete and a new episode begins. The simulator state at each time step is logged; the log file is parsed to extract key performance indicators for analysis.

Initially, the controller is configured with a set of tunable parameters chosen by educated guesswork (which represents what an engineer using the algorithm for the first time might do, in the absence of any other information). Then, in each successive experiment, variations in each

of the tunable parameters are tested, to determine both the optimal value for the parameter, and how susceptible the algorithm is to variations in the parameter. Eventually, this allows the selection of an optimal parameter set.

### 6.1.2 Performance Metrics

The metrics used for evaluating the performance of the Q-Learning based controller are essentially the same as those used for the PID controller, as detailed in §5.1.2: the primary metric is the length of each episode, as it captures both elements of stability and response speed.

In contrast to the PID controller case, however, because the controller uses a machine learning algorithm, it is expected to initially perform poorly, but for the performance to improve over the course of each simulation. Accordingly, the transient response of the episode length must be considered, in addition to the steady-state mean value at the end of the trial. The transient response reflects how quickly the controller is able to learn, which is of significant importance, since in many control systems, it is unacceptable to have the plant behaviour be unstable for any significant period of time.

### 6.1.3 The Term ‘Stability’

The term stability is typically used in reference to control system design to indicate that the time-domain response to a particular input remains bounded. In any practical control system it is generally desired that oscillations occurring in response to a disturbance decay to zero (or at the very least remain constant in amplitude), rather than increasing without bound. This remains true in the design of a reinforcement learning based control system. However, in the field of machine learning, the term is used to describe the long-term behaviour of the learning process.

The performance of the reinforcement learning controller is expected to improve with time, as the state space of the environment is explored, and better policies discovered. However, it will most likely take a reasonable length of time for the *entire* state space to be *comprehensively* explored, and for an optimal policy to be found. This means that even if the performance in a particular episode is considered good, the performance in the subsequent episode may be very poor, if the initial conditions of the episode place the agent into a region of the state space which has not been explored as well. Hence, the performance of the controller may not increase monotonically.

In this context, the term ‘stability’ is used to refer to how much variation in performance the controller displays, after the median performance of the controller has visibly settled to a steady state. The two usages of the term are related: large increases in episode length are most likely due to the time-domain response of the controller becoming unstable (though limited by boundary conditions) for some finite period of time, then stabilizing again. Even if the median behaviour of the controller in the time domain is stable, the presence of unusually long episodes suggests that there are some cases where the behaviour is not stable, and hence, the overall learning behaviour of the control system is considered to have reduced stability.



## 6.2 Tuning Parameters

### 6.2.1 Initial Baseline

As outlined in §6.1.1, initially a set of parameter values are selected to configure the controller, by educated guesswork; generally values were selected which were similar to those used in example code supplied with the Teachingbox library (despite these examples being intended for totally different environments). The set of initial values selected is shown in Table 6.1.

Note that, in this initial set of parameters, individual values for  $K_{\dot{\theta}}$  are used for the controller for each rotation axis. This was selected because the range of motion available in the yaw axis is greater than in the pitch and roll axes (as detailed in §4.2.2), and so a reward function which penalizes high angular velocities might be more important.

A set of three trials were then performed, using the controller configured with this set of initial parameters. The resulting episode lengths for each trial are plotted in Figure 6.1. As expected, the Q-Learning controller displays ‘learning’ behaviour: the performance is poor at the start of each trial (yielding high values for episode length), but improves rapidly. Eventually, the performance settles to some steady state (in the long term, there is still short term variation due to the randomly selected initial conditions of each episode).

Note that Figure 6.1 (and also all the similar figures in this chapter) is plotted against a logarithmic scale on the vertical axis. Additionally, the maximum value shown on the vertical axis is 5,000 (steps per episode); this may result in the first couple of data points not being shown on the chart. This arrangement is necessary to allow visualisation of both the early-stage learning behaviour and late-stage steady-state performance on a single figure, and to allow easy comparison between experiments.

As was the case in analysing the results of testing the conventional PID controller, it would be preferable to use some aggregate of the three trials for further analysis. However, unlike the PID case, simply concatenating the episode lengths together will be insufficient, because the Q-Learning controller’s performance does not remain constant throughout the experiment. An alternative approach is required.

First, because each trial consists of a different number of episodes (because the number of steps and hence the integral of episode length is constant across trials), the data sets are cropped to the length of the trial with the least number of total episodes. This throws away some data regarding the best performing trial, but the amount of data is too small to be significant, and this makes analysis substantially easier. Then, for each episode number, the median episode length amongst the three trials is selected. As shown in Figure 6.1, the median value is clearly representative of the long term performance characteristics, but fails to adequately capture how stable the controller’s performance is. These medians are then smoothed using a simple low pass

Parameter	Value
$\epsilon$	0.05
$\alpha$	0.2
$\gamma$	0.9
$\lambda$	0.8
$\sigma_{\theta}$	128
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	{0.2, 1.0, 0.2}

Table 6.1: Initial set of tunable parameters for QL controller.

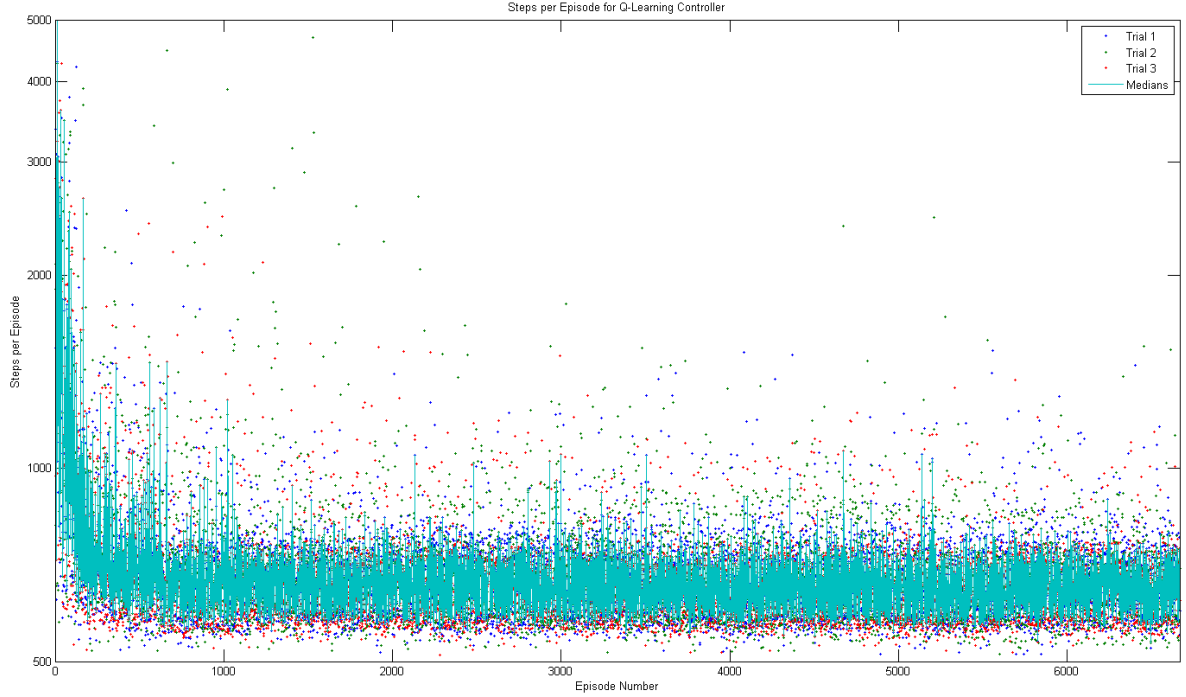


Figure 6.1: Individual episode lengths for QL controller (initial baseline).

filter: the data is filtered by an exponentially weighted moving average (EWMA) filter operating forwards through the data set, and by a similar filter operating backwards through the data set. The results from the two filters are then averaged to obtain smooth filtered representation of the median episode length throughout the experiment.

Figure 6.2 shows the median episode lengths from the experiment (blue) and the smoothed lengths (magenta). The median of the final one thousand data points in the smoothed data set is then calculated; this can be used as a representation of the final steady-state performance of the controller, and is shown as a horizontal cyan line. Also shown (as red horizontal lines) are the median and upper quartile values of the episode length performance of the critically-damped PID controller tested in §5.2.2, which can be used as a reference to gauge the performance of the Q-Learning controller.

Next, the data set is divided into a number of regions: the first region ends when the smoothed median episode length settles to within 30% of the final steady-state value (a vertical cyan line). The second region ends when the smoothed median episode length settles to within 10% of the final steady state value (a second vertical cyan line). Finally, the third and fourth regions each comprise half of the remaining data set (a yellow vertical line).

Now, within each of the four regions, data points from each individual trial are concatenated together. This allows the calculation of some statistics regarding episode lengths, whilst still capturing the learning behaviour exhibited by the controller: the episode numbers at which the edges of the first and second regions occur are indicative of how quickly the learner converges to a steady state; the median steady-state value and total number of episodes are indicative of how good the control performance of the learner becomes; whilst the statistical distribution of episode lengths within each region is indicative of how stable the behaviour of the controller is whilst it learns.

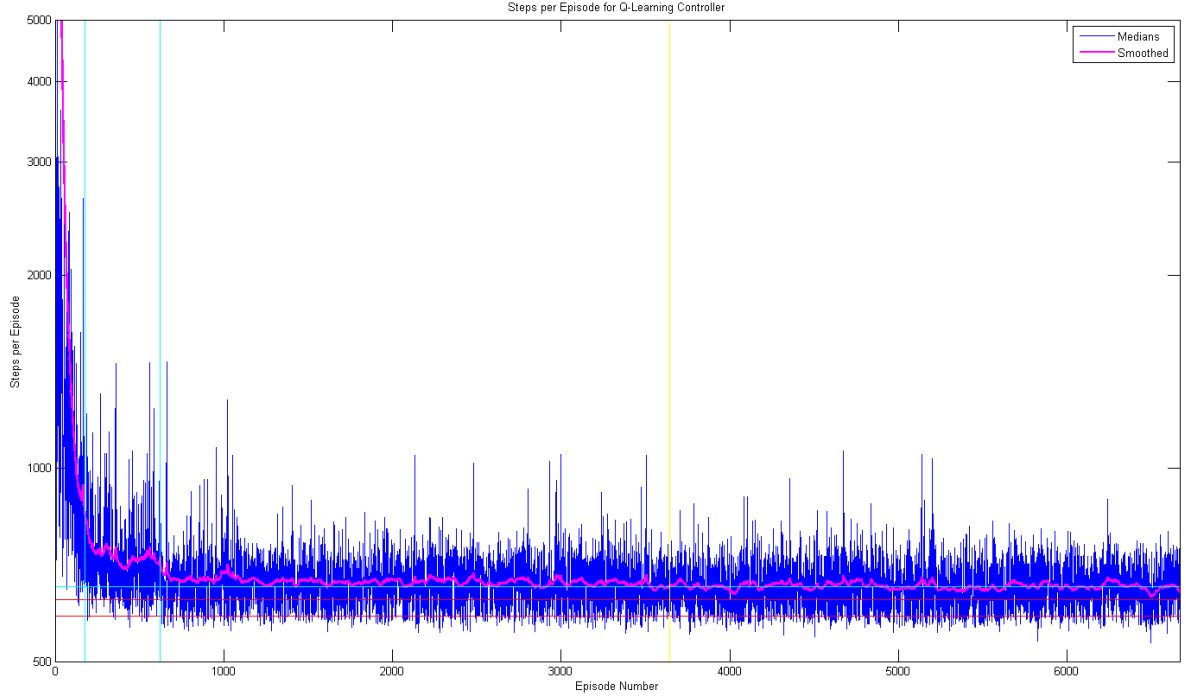


Figure 6.2: Smoothed episode lengths for QL controller (initial baseline).

Figures 6.3, 6.4, 6.5, and 6.6 show the distribution of episode lengths in each region of the experiment, for the individual trials, and also the median and concatenated data sets in that region. It can be seen that using the median values directly would not yield a very accurate representation of the complete distribution of episode lengths within a region, but using the concatenated values will. Hence, the concatenated data set for each region will be used to determine representative statistics for the performance of the experiment overall.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.7. The learning behaviour is clearly visible. The Key Performance Indicators (KPIs) derived from the experiment are also shown in Table 6.2.

As illustrated in Figure 6.2, the median performance of the Q-Learning controller after five

<b>Overall Performance</b>				
Total Length (Episodes)	6667			
Steady State Median Length (Steps)	654			
	<b>Region 1</b>	<b>Region 2</b>	<b>Region 3</b>	<b>Region 4</b>
Length (Episodes)	176	444	3024	3023
<b>Episode Length (Steps)</b>				
Mean	1872	867	689	669
Median	1002	698	663	651
Standard Deviation	4438	2185	178	107
Lower Quartile	729	628	597	591
Upper Quartile	1598	842	727	718
Minimum	532	520	512	513
Maximum	68588	71568	6798	2461

Table 6.2: KPIs for QL controller (initial baseline).

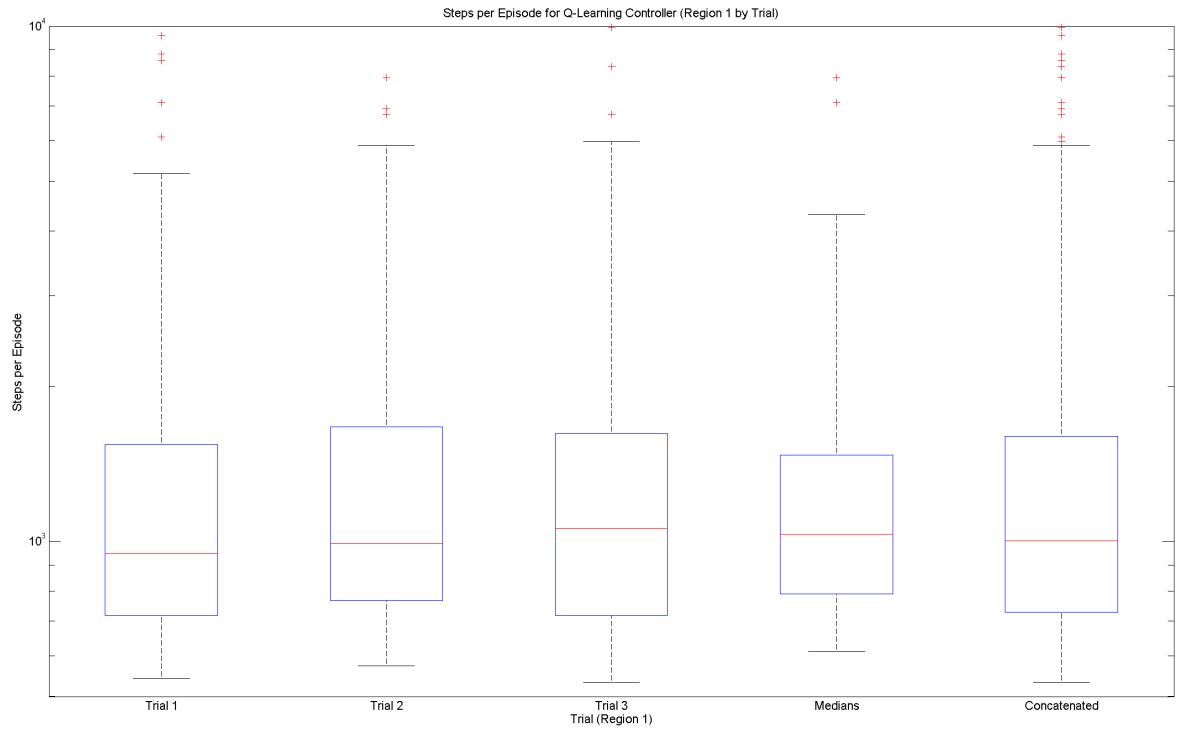


Figure 6.3: Distribution of episode lengths in Region 1 for QL controller (initial baseline).

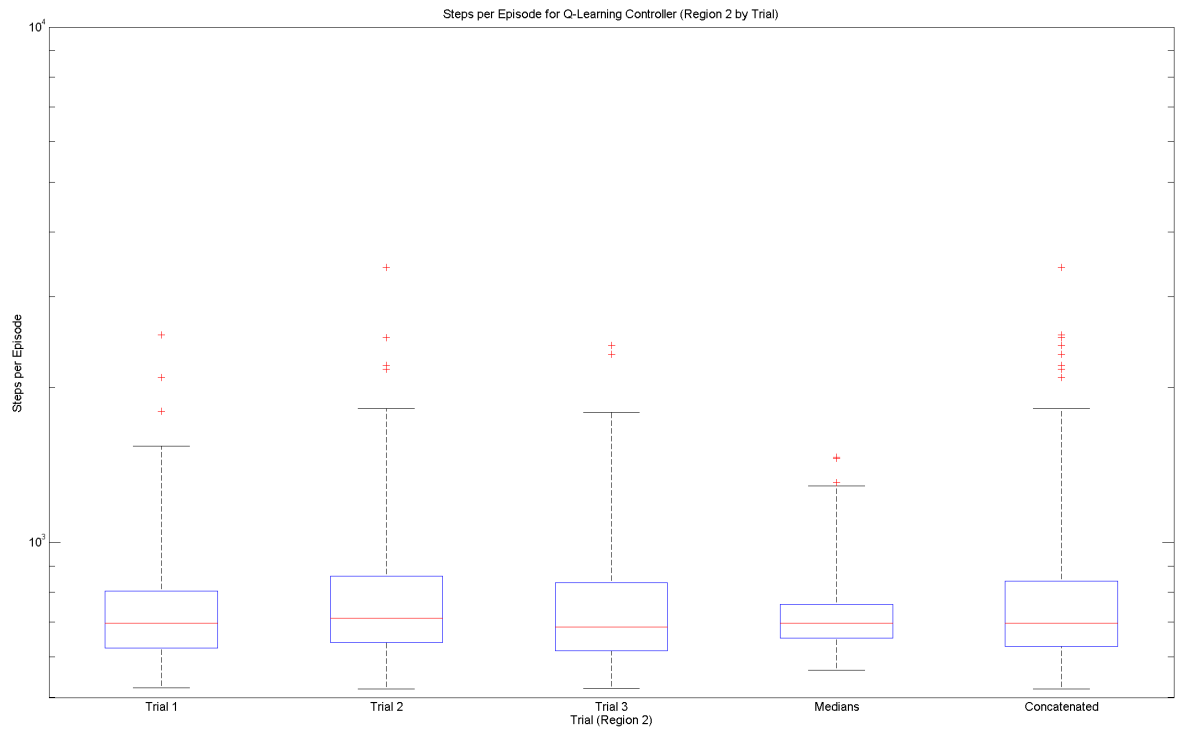


Figure 6.4: Distribution of episode lengths in Region 2 for QL controller (initial baseline).

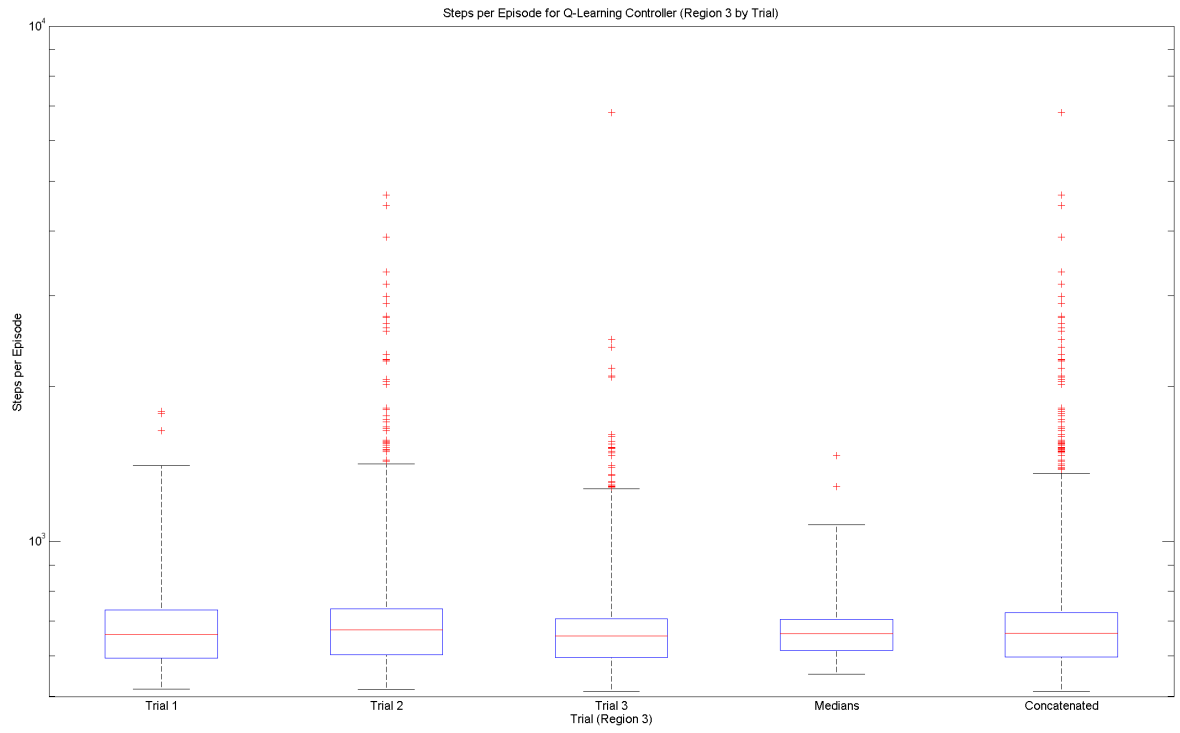


Figure 6.5: Distribution of episode lengths in Region 3 for QL controller (initial baseline).

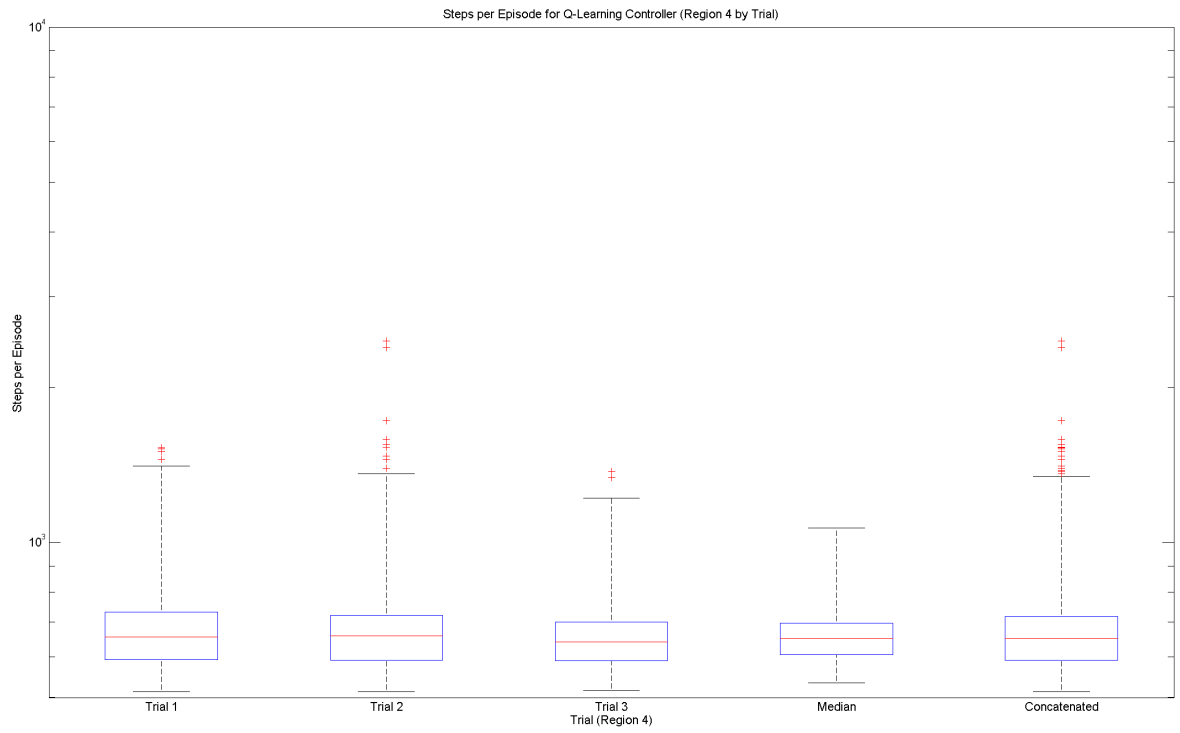


Figure 6.6: Distribution of episode lengths in Region 4 for QL controller (initial baseline).

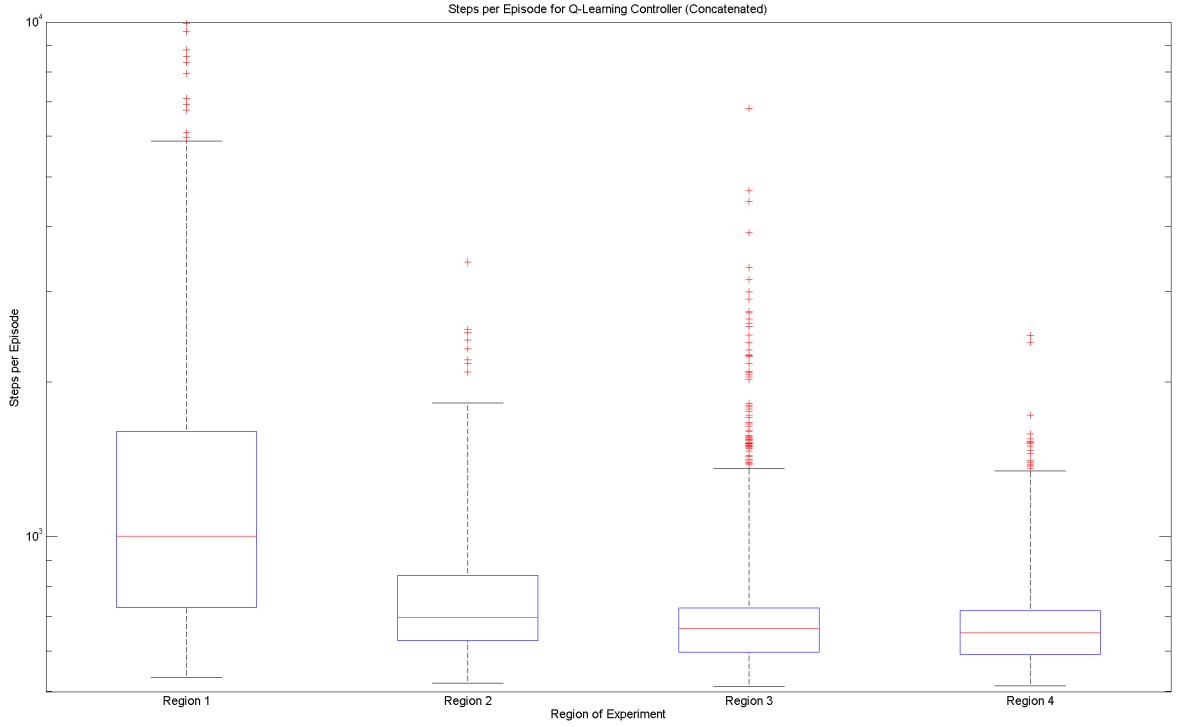


Figure 6.7: Distribution of concatenated episode lengths for QL controller (initial baseline).

million steps is poorer than that of the well tuned PID controller used as a baseline for comparison (median of 654 vs 589). More significantly, the performance of the Q-Learning algorithm is significantly less reliable than the PID controller, as indicated by the considerably greater standard deviation in Region 4 (standard deviation of 107 vs 31). However, it is worth noting that whilst the median episode length does not change significantly between Region 3 and Region 4, the standard deviation is reduced significantly, which suggests that whilst the median performance may only improve marginally with further learning, the reliability of the controller may still improve considerably.

Finally, consideration is also given to the time-domain response of the controller. A sample plot of the response is shown in Figure 6.8: a set of data points were selected randomly from near the end of Region 4, so as to capture the steady-state learned behaviour. The division of the response into individual episodes is clear; the simulator selects a random initial position which appears as a step input, the controller corrects the error, and once the error has remained sufficiently small for at least 500 steps, a new episode begins. Clearly, the responses in the X and Z axes (pitch and roll) are underdamped, whilst the response in the Y axis (yaw) is overdamped for large errors. Whilst not critically damped, the response is not unrecognisably different to that of a PID controller.

### 6.2.2 Varying Learning Rate $\alpha$

A set of experiments were then conducted to characterise the effect which variations in the learner parameter  $\alpha$  would have on the performance of the controller. In each experiment, this parameter was adjusted, whilst the remainder of the parameters remained fixed to values used

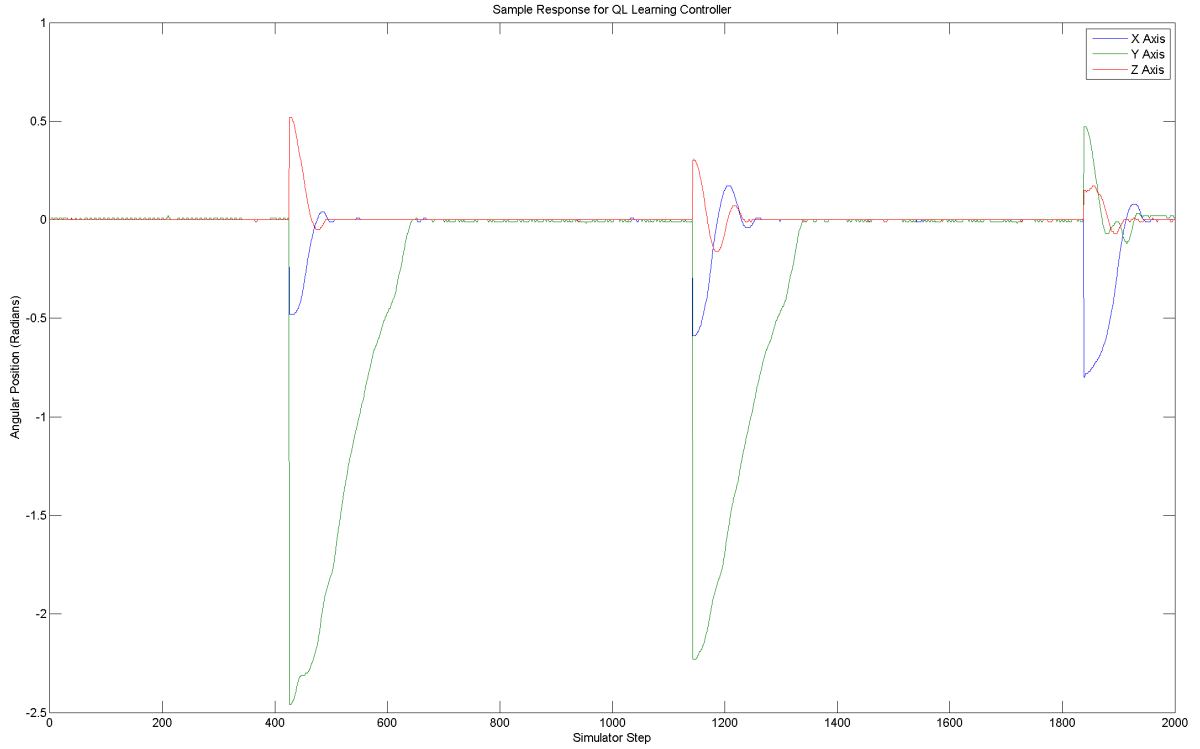


Figure 6.8: Sample time-domain response for QL controller (initial baseline).

previously, to allow comparison with previous experiments.

**High Alpha ( $\alpha = 0.5$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.3; the value of  $\alpha$  (the learning rate) is increased to 0.5. The resulting smoothed set of episode lengths is shown in Figure 6.9.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.10. The key performance characteristics derived from the experiment are also shown in Table 6.4.

Comparison of these key performance values with those of the initial baseline experiment suggests that the increase in  $\alpha$  value results in significantly reduced learning time (represented by the reduction in lengths of Regions 1 and 2, which are indicative of settling time). This would be expected, since  $\alpha$  represents the ‘learning rate’ of the controller. However, this is offset by a substantial increase in the standard deviation of episode lengths later in the experiment. The

Parameter	Value
$\epsilon$	0.05
$\alpha$	0.5
$\gamma$	0.9
$\lambda$	0.8
$\sigma_\theta$	128
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	{0.2, 1.0, 0.2}

Table 6.3: Tunable parameters for QL controller ( $\alpha = 0.5$ ).

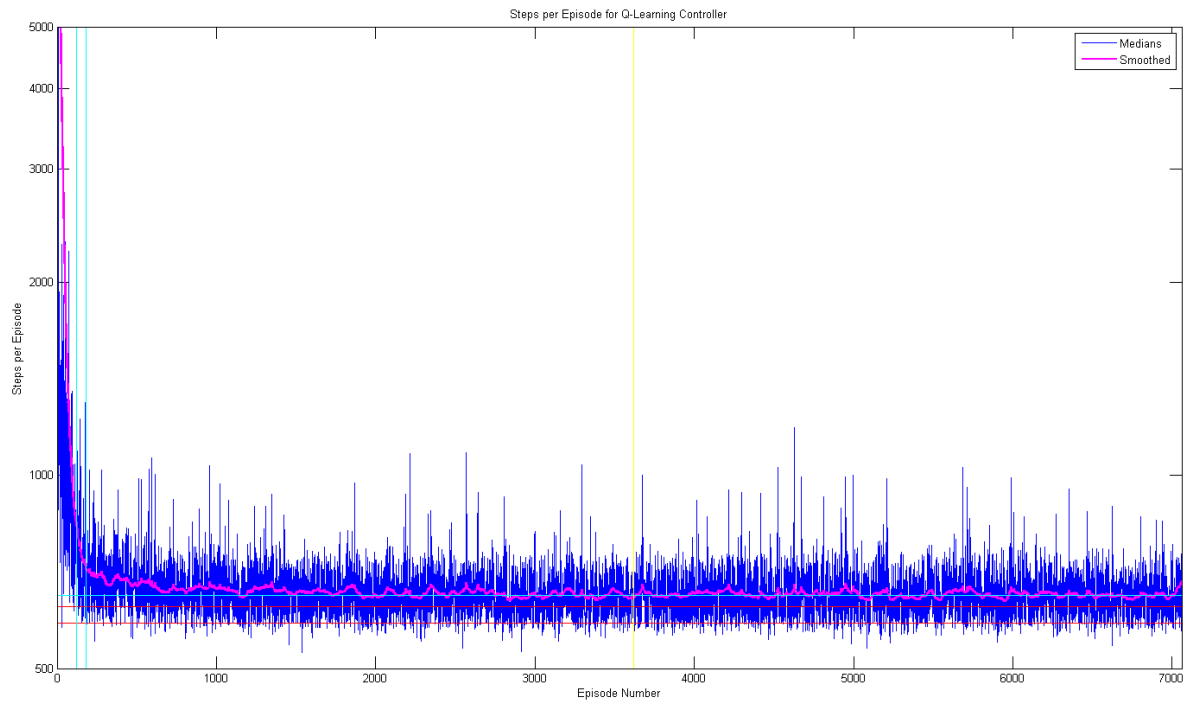


Figure 6.9: Smoothed episode lengths for QL controller ( $\alpha = 0.5$ ).

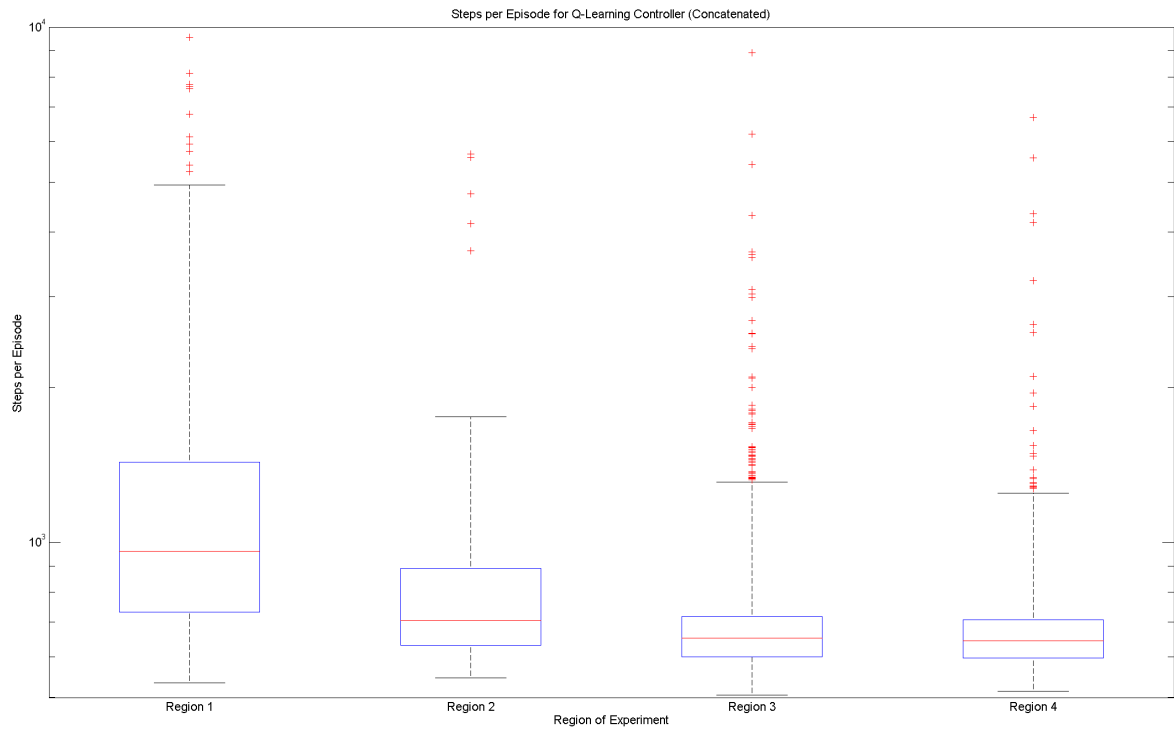


Figure 6.10: Distribution of concatenated episode lengths for QL controller ( $\alpha = 0.5$ ).



<b>Overall Performance</b>				
Total Length (Episodes)	7060			
Steady State Median Length (Steps)	650			
	Region 1	Region 2	Region 3	Region 4
Length (Episodes)	122	60	3439	3439
<b>Episode Length (Steps)</b>				
Mean	1670	970	685	672
Median	961	707	652	644
Standard Deviation	2703	1246	275	316
Lower Quartile	732	635	599	596
Upper Quartile	1433	901	718	708
Minimum	534	546	505	514
Maximum	26805	14816	17797	27453

Table 6.4: KPIs for QL controller ( $\alpha = 0.5$ ).

steady-state median episode length is largely unchanged, whilst the reduced learning time results in a small improvement in terms of total episodes completed.

Finally, consideration is also given to the time-domain response of the controller. A sample plot of the response is shown in Figure 6.11. The response of the controller remains similar to that of the initial baseline, with underdamping present in the pitch and roll axes, and severe overdamping in the yaw axis for large errors.

**Low Alpha ( $\alpha = 0.1$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.5; the value of  $\alpha$  (the learning rate) is decreased to 0.1. The resulting smoothed set of episode lengths is shown in Figure 6.12.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.13. The key performance characteristics derived from the experiment are also shown in Table 6.6.

Comparison of these key performance values with those of the initial baseline experiment suggests that the reduction in  $\alpha$  value results in significantly increased learning time (represented by the increase in lengths of Regions 1 and 2, which are indicative of settling time). This would be expected, since  $\alpha$  represents the ‘learning rate’ of the controller. However, this is offset by a substantial reduction in the standard deviation of episode lengths throughout the experiment. The steady-state median episode length is largely unchanged, but the increased learning time results in a small reduction in terms of total episodes completed.

Finally, consideration is also given to the time-domain response of the controller. A sample plot of the response is shown in Figure 6.14. The response of the controller remains similar to that of the initial baseline, with underdamping present in the pitch and roll axes, and severe overdamping in the yaw axis for large errors.

Parameter	Value
$\epsilon$	0.05
$\alpha$	0.1
$\gamma$	0.9
$\lambda$	0.8
$\sigma_\theta$	128
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	{0.2, 1.0, 0.2}

Table 6.5: Tunable parameters for QL controller ( $\alpha = 0.1$ ).

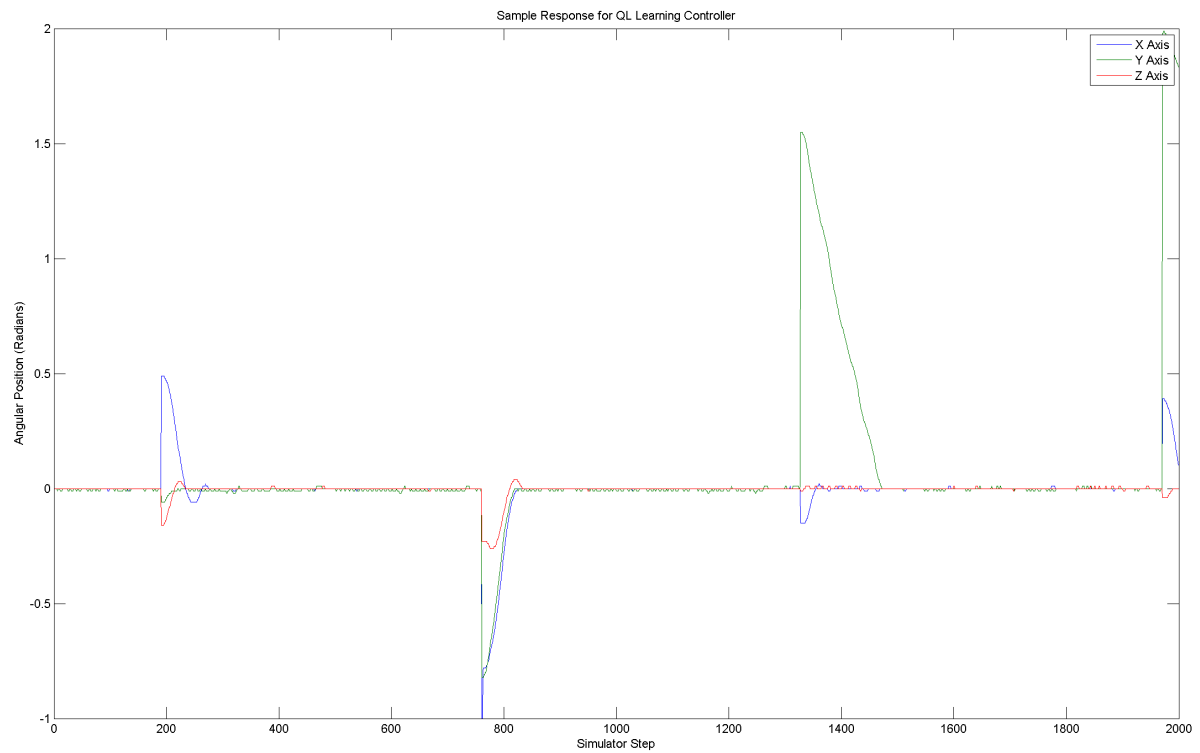


Figure 6.11: Sample time-domain response for QL controller ( $\alpha = 0.5$ ).

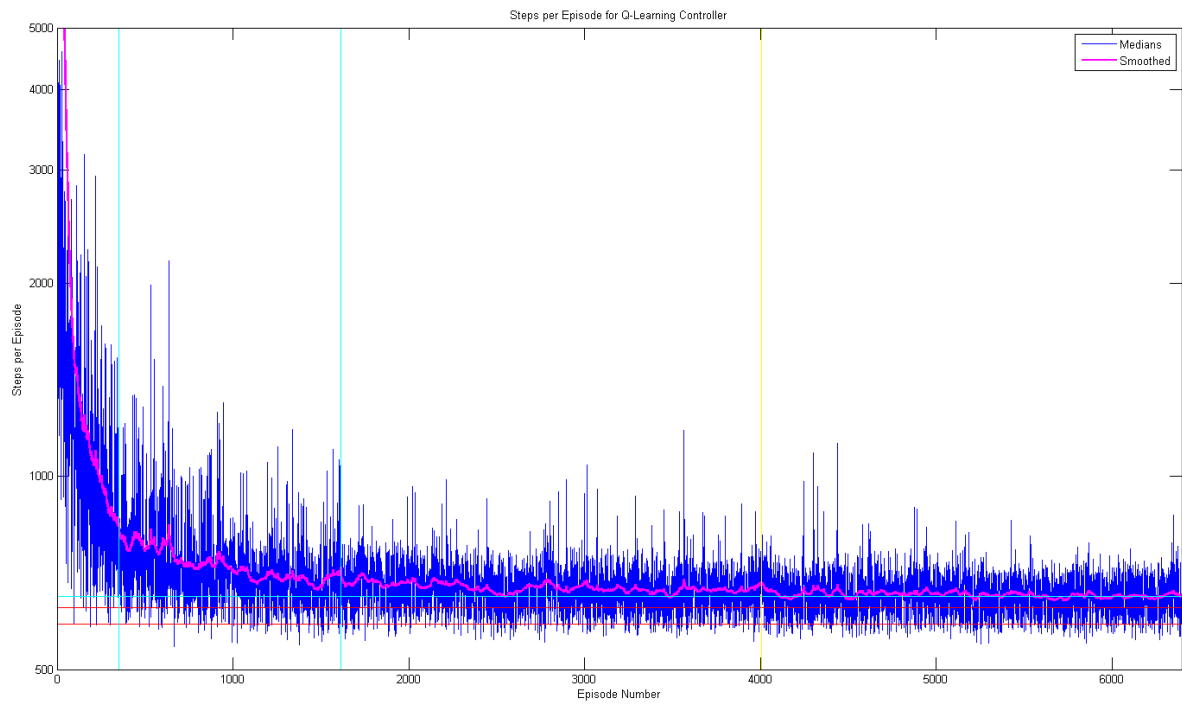


Figure 6.12: Smoothed episode lengths for QL controller ( $\alpha = 0.1$ ).

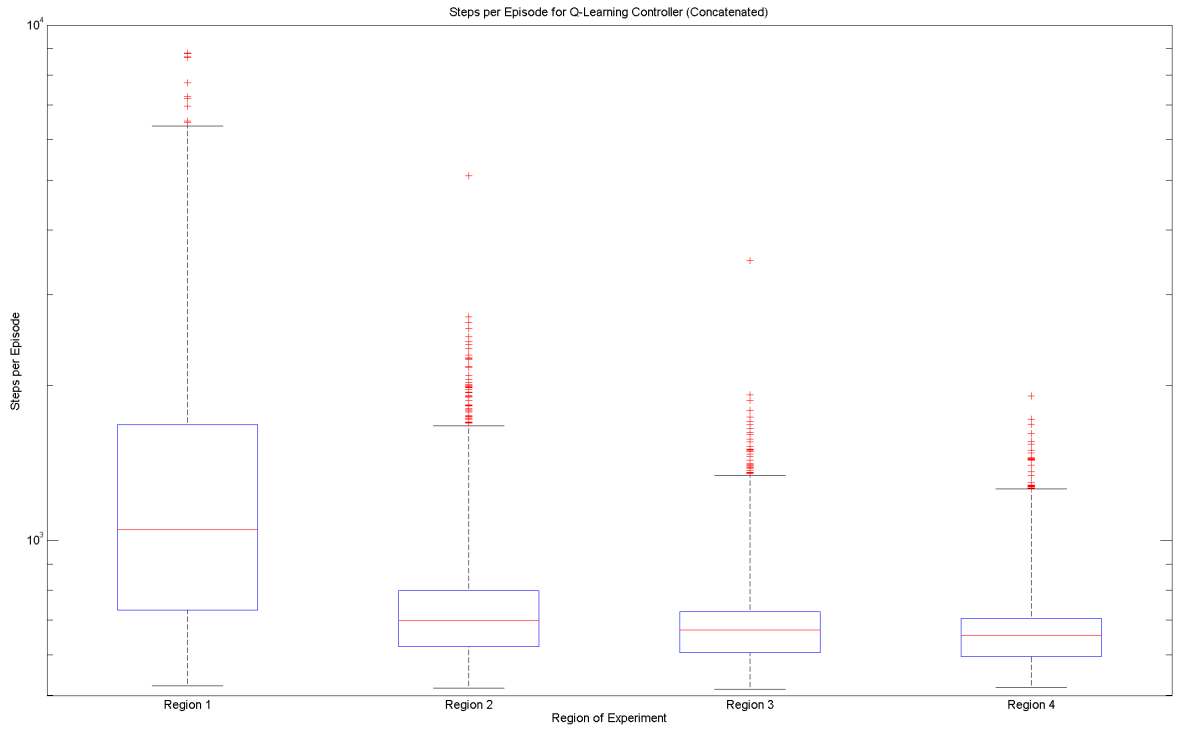


Figure 6.13: Distribution of concatenated episode lengths for QL controller ( $\alpha = 0.1$ ).

<b>Overall Performance</b>				
Total Length (Episodes)	6397			
Steady State Median Length (Steps)	651			
	Region 1	Region 2	Region 3	Region 4
Length (Episodes)	350	1262	2393	2392
<b>Episode Length (Steps)</b>				
Mean	1752	774	689	667
Median	1051	699	670	653
Standard Deviation	3852	510	131	104
Lower Quartile	732	632	605	595
Upper Quartile	1676	799	727	706
Minimum	522	517	514	518
Maximum	67498	28051	3496	1905

Table 6.6: KPIs for QL controller ( $\alpha = 0.1$ ).

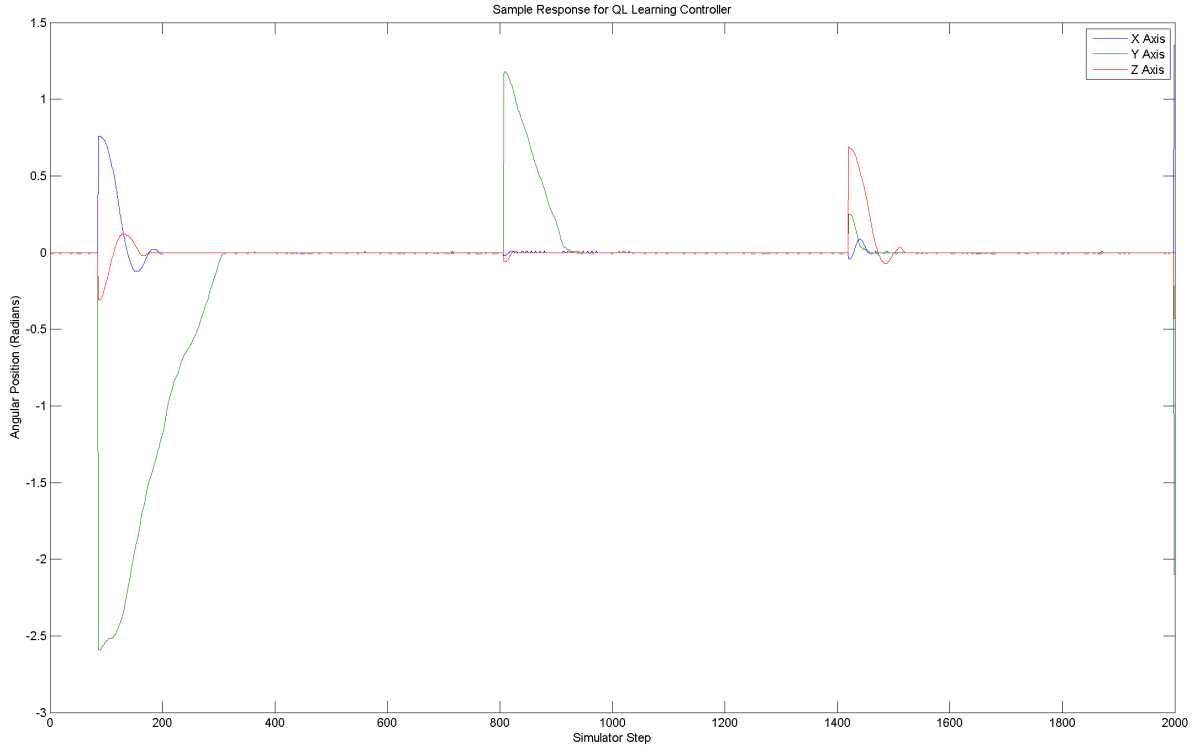


Figure 6.14: Sample time-domain response for QL controller ( $\alpha = 0.1$ ).

The experiments show that there is a clear trade-off in selecting a value for  $\alpha$ , between faster learning, and more stable steady-state which is resistant to noise and local minima. However, the steady-state median length and time-domain response of the controller remains constant regardless of changes to  $\alpha$ ; this is expected, since  $\alpha$  only characterises the learning process, not what is learned by the controller.

The original value of  $\alpha = 0.2$  seems like a suitable compromise between speed and stability, so further experiments will continue to use this value.

### 6.2.3 Varying Discount Rate $\gamma$

A set of experiments were then conducted to characterise the effect which variations in the learner parameter  $\gamma$  would have on the performance of the controller. In each experiment, this parameter was adjusted, whilst the remainder of the parameters remained fixed to values used previously, to allow comparison with previous experiments.

**High Gamma ( $\gamma = 0.95$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.7; the value of  $\gamma$  (the discount rate) is increased to 0.95. The resulting smoothed set of episode lengths is shown in Figure 6.15.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.16. The key performance characteristics derived from the experiment are also shown in Table 6.8.

Parameter	Value
$\epsilon$	0.05
$\alpha$	0.2
$\gamma$	0.95
$\lambda$	0.8
$\sigma_\theta$	128
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	$\{0.2, 1.0, 0.2\}$

Table 6.7: Tunable parameters for QL controller ( $\gamma = 0.95$ ).

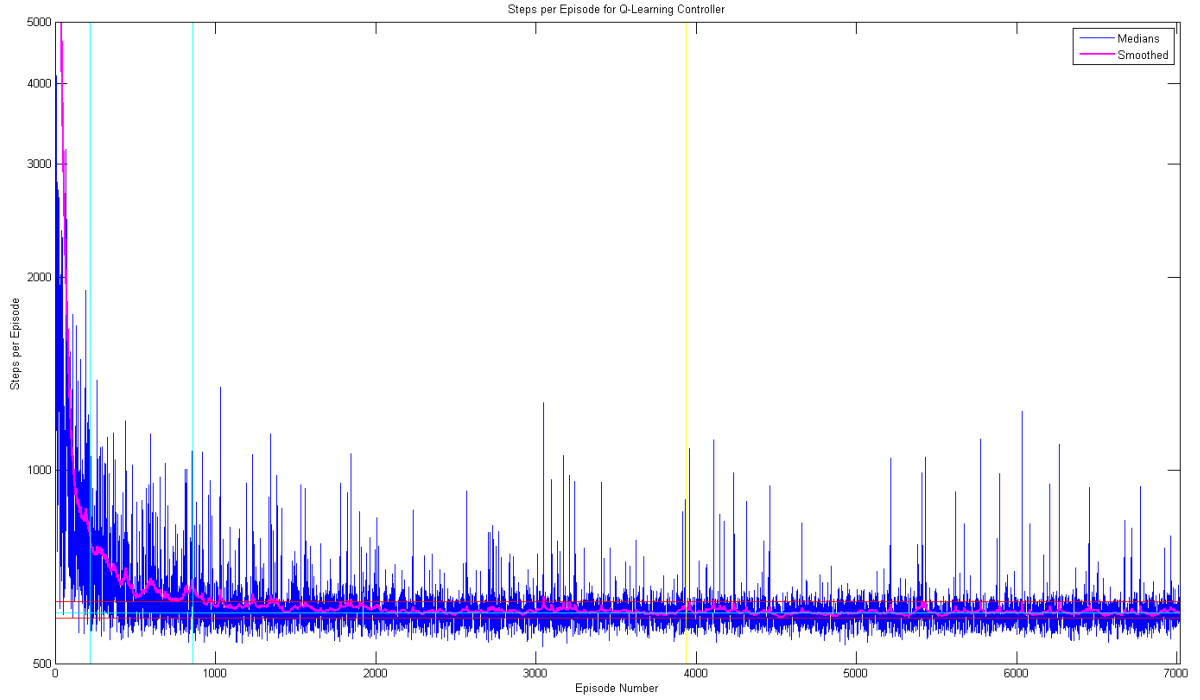


Figure 6.15: Smoothed episode lengths for QL controller ( $\gamma = 0.95$ ).

<b>Overall Performance</b>				
Total Length (Episodes)	7017			
Steady State Median Length (Steps)	600			
	Region 1	Region 2	Region 3	Region 4
Length (Episodes)	220	639	3079	3079
<b>Episode Length (Steps)</b>				
Mean	1407	720	648	631
Median	929	631	598	596
Standard Deviation	2801	302	233	186
Lower Quartile	680	582	571	571
Upper Quartile	1392	747	632	626
Minimum	530	523	516	517
Maximum	46319	9134	5747	4290

Table 6.8: KPIs for QL controller ( $\gamma = 0.95$ ).

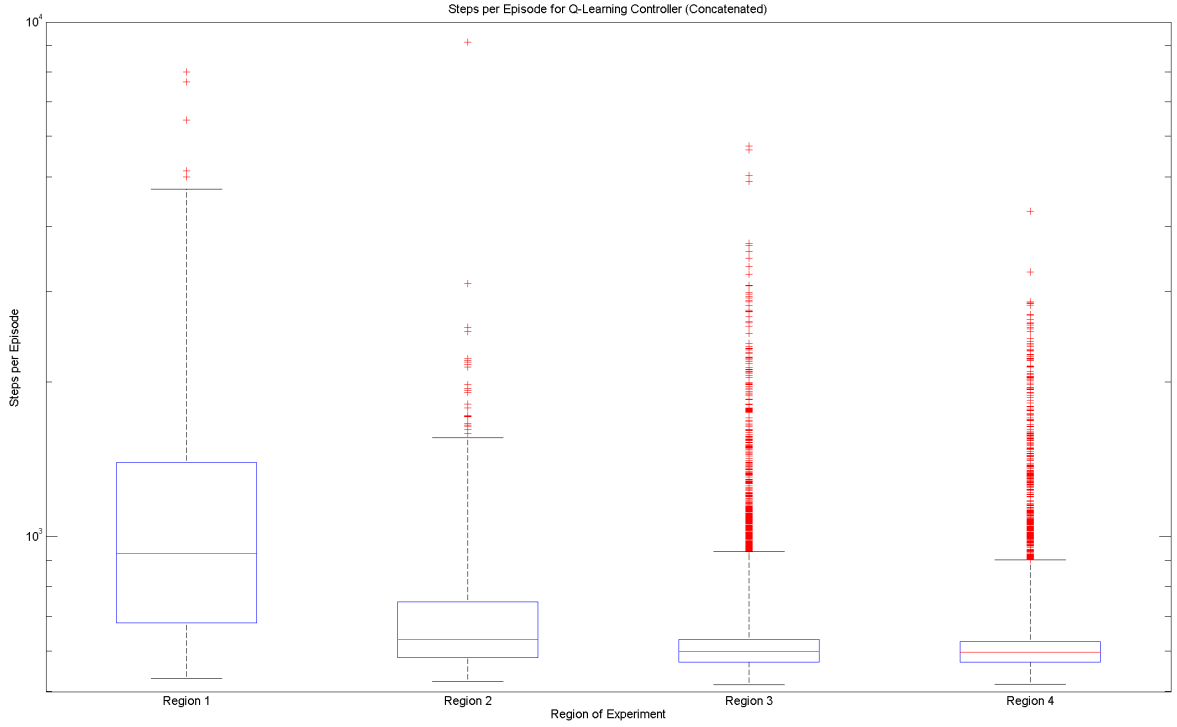


Figure 6.16: Distribution of concatenated episode lengths for QL controller ( $\gamma = 0.95$ ).

Comparison of these key performance values with those of the initial baseline experiment suggests that this increase in  $\gamma$  value results in an improvement to the median episode length, throughout each region of the experiment. The median episode length in this experiment is less than the upper quartile of episode lengths recorded by the conventional PID controller, although still greater than the median of episode lengths delivered by the PID controller. The learning performance is not significantly affected, as the regions are approximately the same length as during the baseline experiment. One interesting characteristic is that the overall standard deviation of episode lengths is better than the baseline for regions one and two, but not as good in regions three and four. Visual comparison of Figures 6.2 and 6.15 suggests that whilst the higher gamma value results in improved median performance, there are a number of outliers with significantly degraded performance. This is also captured by comparison of the mean and median values for both this experiment and the baseline; the proportional difference between mean and median is less in the baseline case. This could suggest that whilst the higher gamma value results in generally improved control behaviour, it may also promote issues with the stability of the resulting controller.

Finally, consideration is also given to the time-domain response of the controller. A sample plot of the response is shown in Figure 6.17. The response of the controller remains similar to that of the initial baseline, with underdamping present in the pitch and roll axes; there is still overdamping in the yaw axis for large errors, though it appears less pronounced.

**High Gamma ( $\gamma = 0.99$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.9; the value of  $\gamma$  (the discount rate) is increased to 0.99. The resulting smoothed set of episode lengths is shown in Figure 6.18.

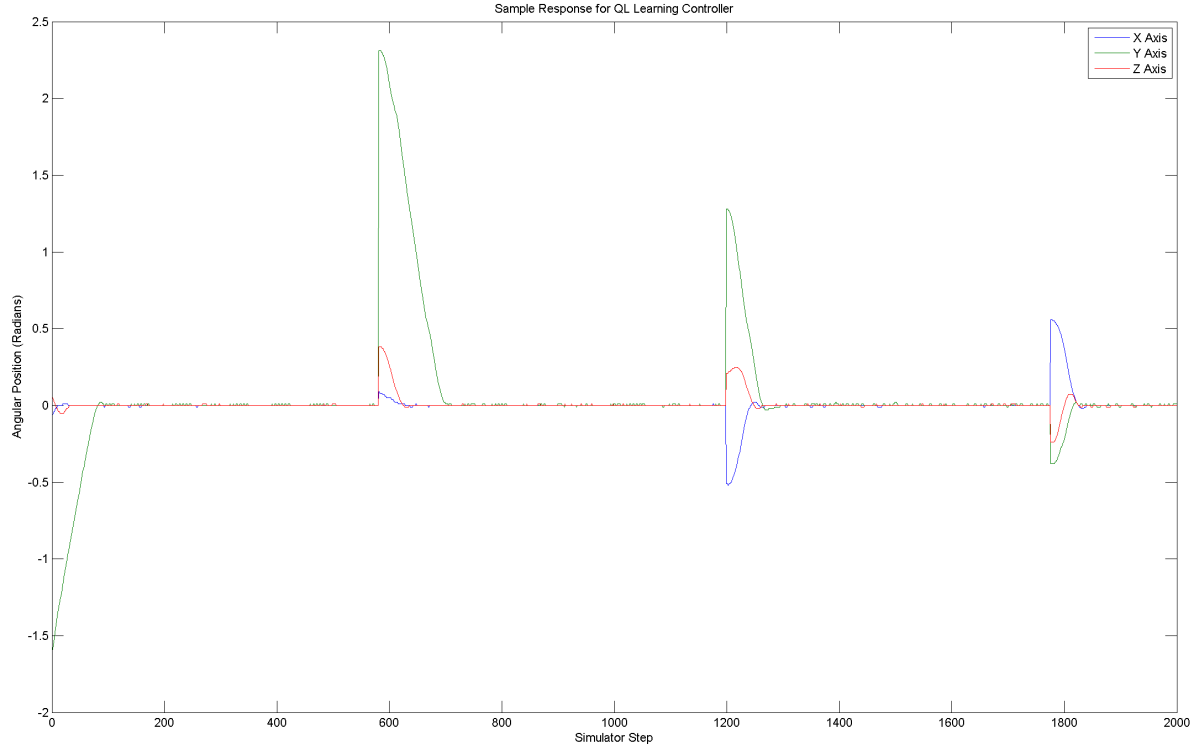


Figure 6.17: Sample time-domain response for QL controller ( $\gamma = 0.95$ ).

Parameter	Value
$\epsilon$	0.05
$\alpha$	0.2
$\gamma$	0.99
$\lambda$	0.8
$\sigma_\theta$	128
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	$\{0.2, 1.0, 0.2\}$

Table 6.9: Tunable parameters for QL controller ( $\gamma = 0.99$ ).

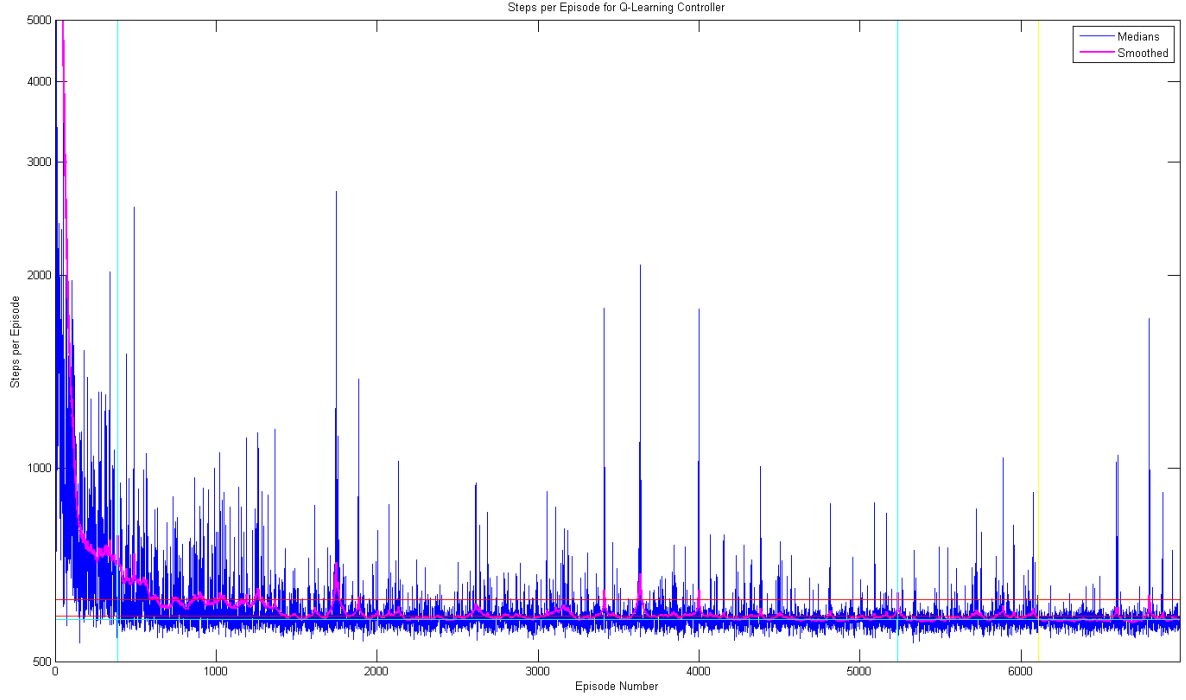


Figure 6.18: Smoothed episode lengths for QL controller ( $\gamma = 0.99$ ).

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.19. The key performance characteristics derived from the experiment are also shown in Table 6.10.

Comparison of these key performance values with those of the initial baseline experiment suggests that the further increase in  $\gamma$  value results in further improvements to median episode length. However, the observations made for the  $\gamma = 0.95$  case regarding instability appear justified; whilst the median episode length is now directly comparable to that of the PID controller, the standard deviation of episode lengths is substantially increased upon the baseline. The greater distribution of episode lengths means the learning behaviour does not settle adequately, which

<b>Overall Performance</b>				
Total Length (Episodes)	6986			
Steady State Median Length (Steps)	581			
	<b>Region 1</b>	<b>Region 2</b>	<b>Region 3</b>	<b>Region 4</b>
Length (Episodes)	390	4845	876	875
<b>Episode Length (Steps)</b>				
Mean	1552	664	643	628
Median	744	582	579	578
Standard Deviation	5959	687	564	301
Lower Quartile	611	565	564	563
Upper Quartile	1106	608	597	594
Minimum	500	511	516	508
Maximum	101455	44633	24916	6890

Table 6.10: KPIs for QL controller ( $\gamma = 0.99$ ).



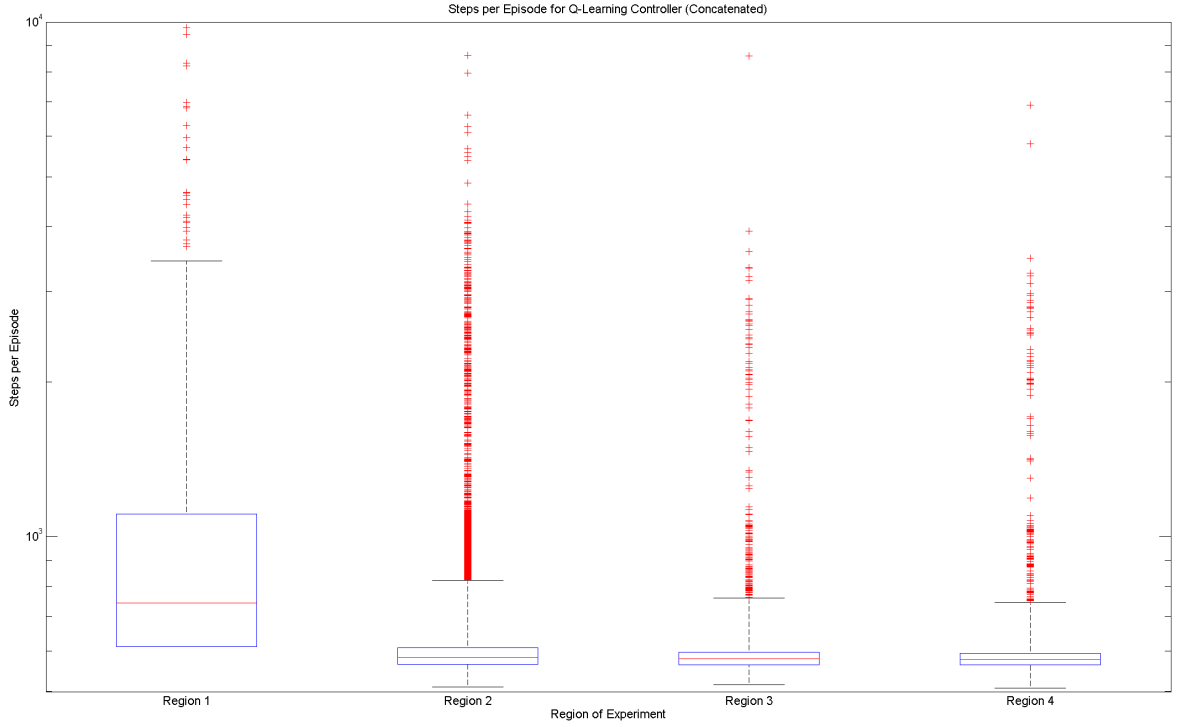


Figure 6.19: Distribution of concatenated episode lengths for QL controller ( $\gamma = 0.99$ ).

results in the length of region two being extended considerably, and that the total number of episodes completed is less than in the  $\gamma = 0.95$  case. This behaviour seems consistent with the behaviour of similar reinforcement learning experiments available in literature (such as by Singh [47]), where increasing discount rate improves performance, until at some high value of gamma, performance degenerates rapidly.

Finally, consideration is also given to the time-domain response of the controller. A sample plot of the response is shown in Figure 6.20. The response of the pitch and roll axes remains generally underdamped, as in previous experiments, however the yaw response is no longer excessively overdamped.

**Low Gamma ( $\gamma = 0.85$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.11; the value of  $\gamma$  (the discount rate) is decreased to 0.85. The resulting smoothed set of episode lengths is shown in Figure 6.21.

Parameter	Value
$\epsilon$	0.05
$\alpha$	0.1
$\gamma$	0.85
$\lambda$	0.8
$\sigma_\theta$	128
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	{0.2, 1.0, 0.2}

Table 6.11: Tunable parameters for QL controller ( $\gamma = 0.85$ ).

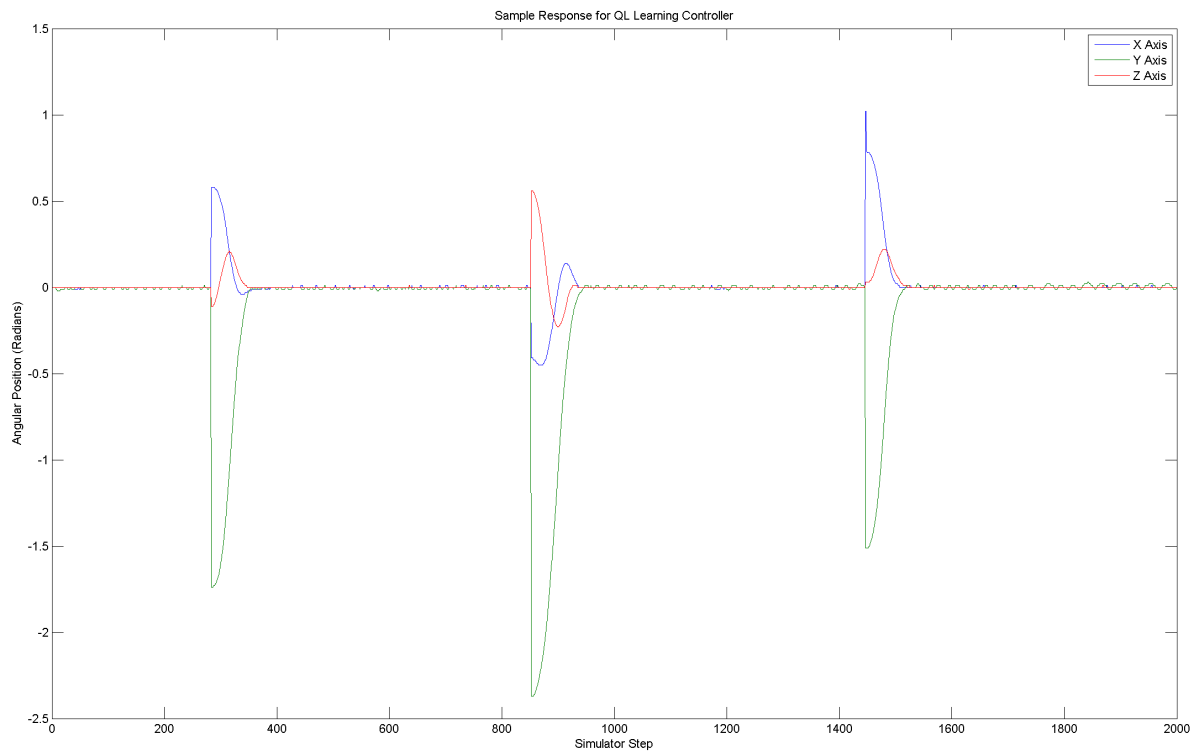


Figure 6.20: Sample time-domain response for QL controller ( $\gamma = 0.99$ ).

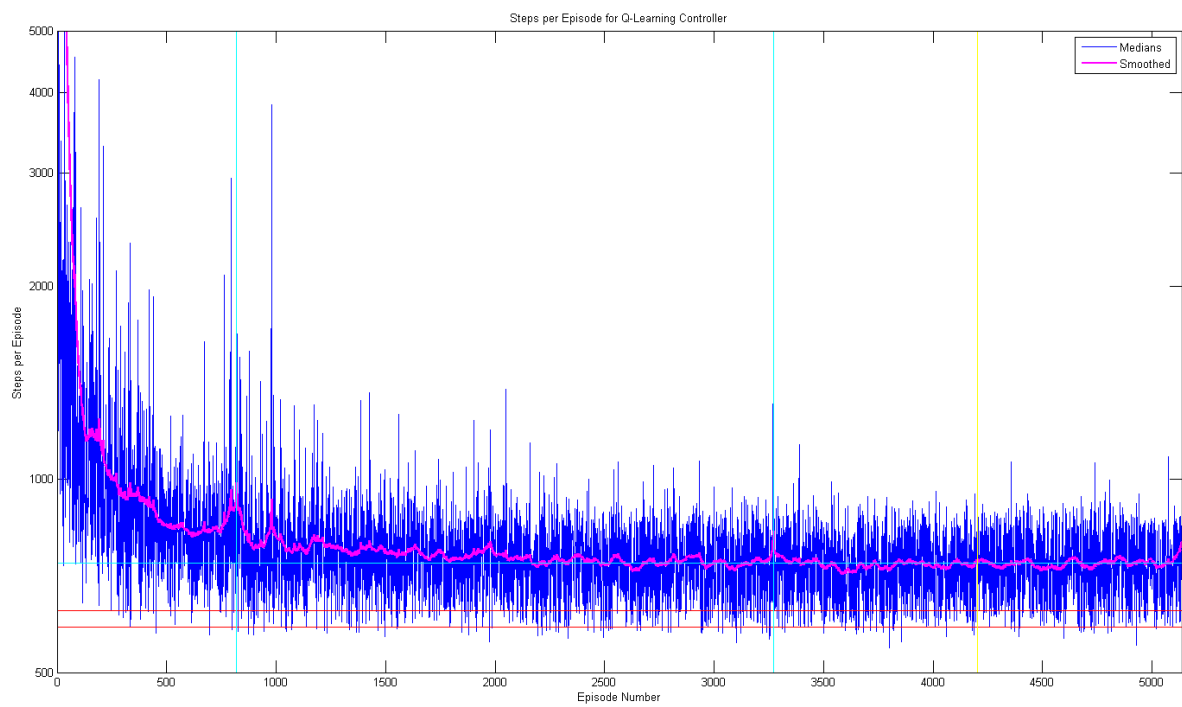


Figure 6.21: Smoothed episode lengths for QL controller ( $\gamma = 0.85$ ).

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.22. The key performance characteristics derived from the experiment are also shown in Table 6.12.

Comparison of these key performance values with those of the initial baseline experiment suggests that the reduction in  $\gamma$  value results in an increase in median episode length (which results in a reduction in total number of episodes completed). There is also an increase in the standard deviation of episode lengths, which additionally results in increased settling time for regions one and two. However, it is worth noting that the number of outliers in Figure 6.21 is reduced from those experiments with higher discount rates, which could result in reduced issues with instability in a controller.

Finally, consideration is also given to the time-domain response of the controller. A sample plot of the response is shown in Figure 6.23. The response of the controller remains similar to that of the initial baseline, with underdamping present in the pitch and roll axes, and severe overdamping in the yaw axis for large errors.

The experiments indicate that there is a clear trade-off in selecting a value for  $\gamma$ , with improved performance (considering both median and standard deviation) carrying a risk of instability. However, it is clear that operating with too low a decay rate will yield unacceptably poor performance for little benefit.

Whilst the value of  $\gamma = 0.95$  appears to offer a performance advantage over the original value of  $\gamma = 0.9$ , without adding much instability, there is risk associated with operating close to the point at which performance will begin to become unstable (as demonstrated at  $\gamma = 0.99$ ). Accordingly, further experiments will continue to use the original value of  $\gamma = 0.9$  to ensure that operating in this unstable region does not affect other experimental results.

#### 6.2.4 Varying Eligibility Trace Decay Rate $\lambda$

A set of experiments were then conducted to characterise the effect which variations in the learner parameter  $\lambda$  would have on the performance of the controller. In each experiment, this parameter was adjusted, whilst the remainder of the parameters remained fixed to values used previously, to allow comparison with previous experiments.

**High Lambda ( $\lambda = 0.9$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.13; the value of  $\lambda$  (the decay rate) is increased to 0.9.

<b>Overall Performance</b>				
Total Length (Episodes)	5136			
Steady State Median Length (Steps)	739			
	<b>Region 1</b>	<b>Region 2</b>	<b>Region 3</b>	<b>Region 4</b>
Length (Episodes)	820	2451	933	932
<b>Episode Length (Steps)</b>				
Mean	1650	852	766	763
Median	885	753	740	734
Standard Deviation	6284	1730	352	306
Lower Quartile	728	643	629	632
Upper Quartile	1152	862	843	841
Minimum	526	513	520	514
Maximum	168472	49800	11314	8999

Table 6.12: KPIs for QL controller ( $\gamma = 0.85$ ).

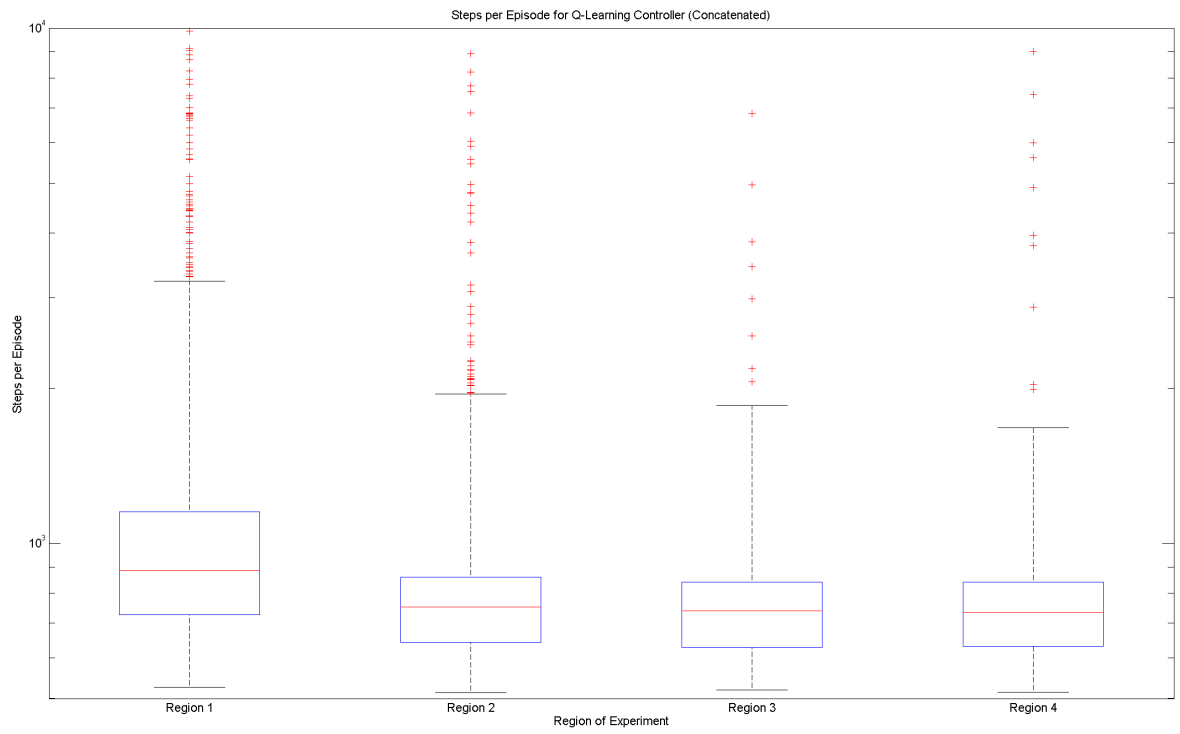


Figure 6.22: Distribution of concatenated episode lengths for QL controller ( $\gamma = 0.85$ ).

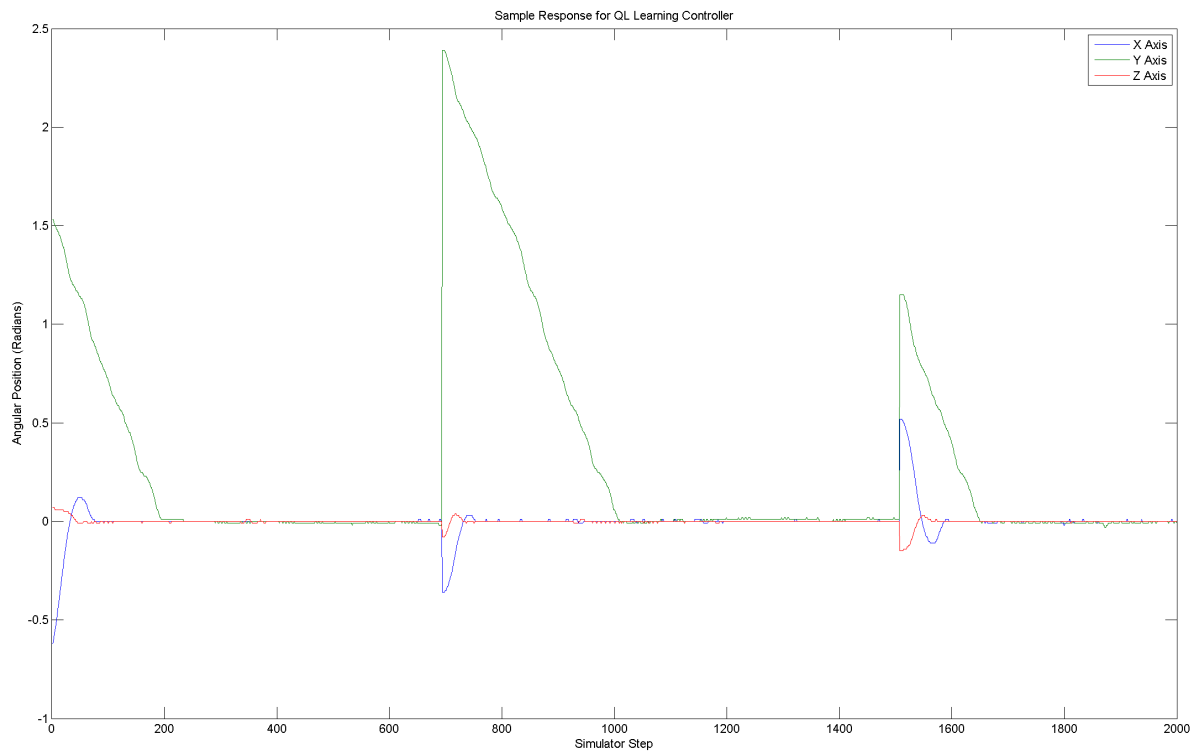


Figure 6.23: Sample time-domain response for QL controller ( $\gamma = 0.85$ ).

The resulting smoothed set of episode lengths is shown in Figure 6.24.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.25. The key performance characteristics derived from the experiment are also shown in Table 6.14.

Comparison of these key performance values with those of the initial baseline experiment suggests that the increase in  $\lambda$  value results in minimal changes to the overall performance of the controller. There is a slight increase in median steady-state episode length. However, there are also notable reductions to the standard deviation of episode lengths during the earlier regions of the experiment, which gives a slight overall improvement in terms of total episodes completed.

Finally, consideration is also given to the time-domain response of the controller. A sample plot of the response is shown in Figure 6.26. The response of the controller remains similar to that of the initial baseline, with underdamping present in the pitch and roll axes; there is still significant overdamping in the yaw axis for large errors.

**High Lambda ( $\lambda = 0.95$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.15; the value of  $\lambda$  (the decay rate) is increased to 0.95. The resulting smoothed set of episode lengths is shown in Figure 6.27.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.28. The key performance characteristics derived from the experiment are also shown in Table 6.16.

Comparison of these key performance values with those of the initial baseline experiment suggests that the increase in  $\lambda$  value results in minimal changes to the overall performance of the controller. The median episode lengths are very similar to those of the baseline experiment. There is a substantial increase in standard deviation of episode length during regions one and two, but the standard deviation in region four is less than that of the baseline.

Finally, consideration is also given to the time-domain response of the controller. A sample plot of the response is shown in Figure 6.29. The response of the controller remains similar to that of the initial baseline, with underdamping present in the pitch and roll axes; there is still significant overdamping in the yaw axis for large errors.

**Low Lambda ( $\lambda = 0.7$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.17; the value of  $\lambda$  (the decay rate) is decreased to 0.7. The resulting smoothed set of episode lengths is shown in Figure 6.30.

Parameter	Value
$\epsilon$	0.05
$\alpha$	0.2
$\gamma$	0.9
$\lambda$	0.9
$\sigma_\theta$	128
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	{0.2, 1.0, 0.2}

Table 6.13: Tunable parameters for QL controller ( $\lambda = 0.9$ ).

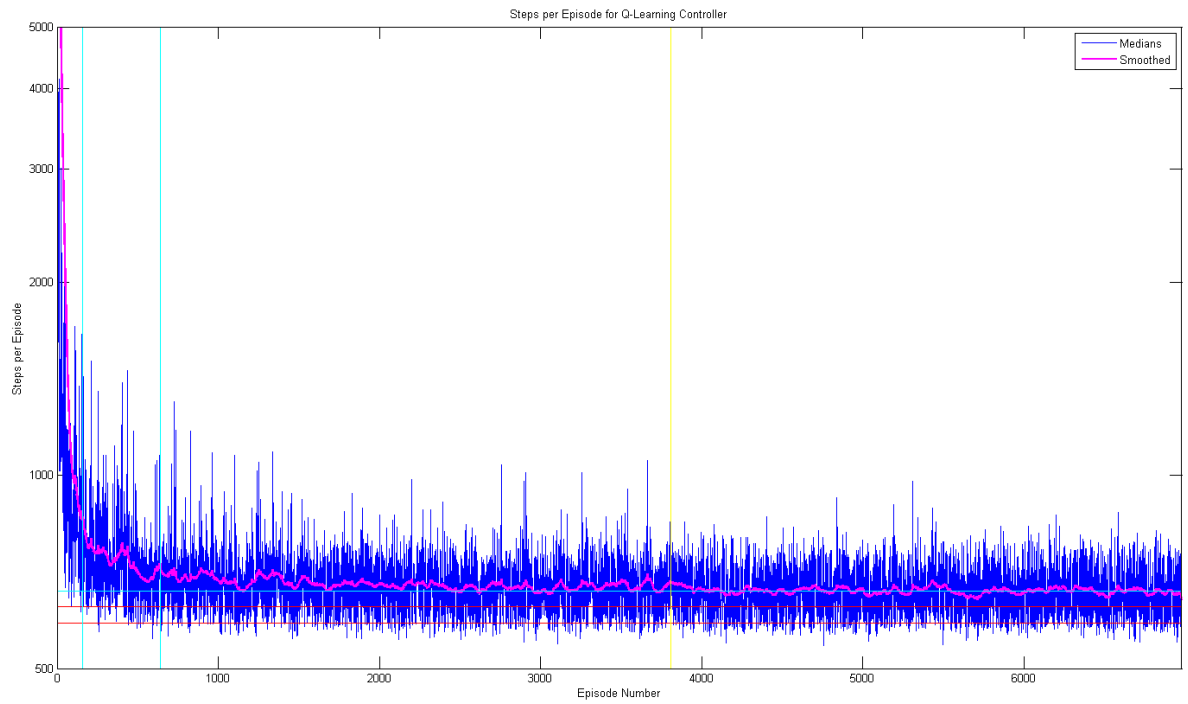


Figure 6.24: Smoothed episode lengths for QL controller ( $\lambda = 0.9$ ).

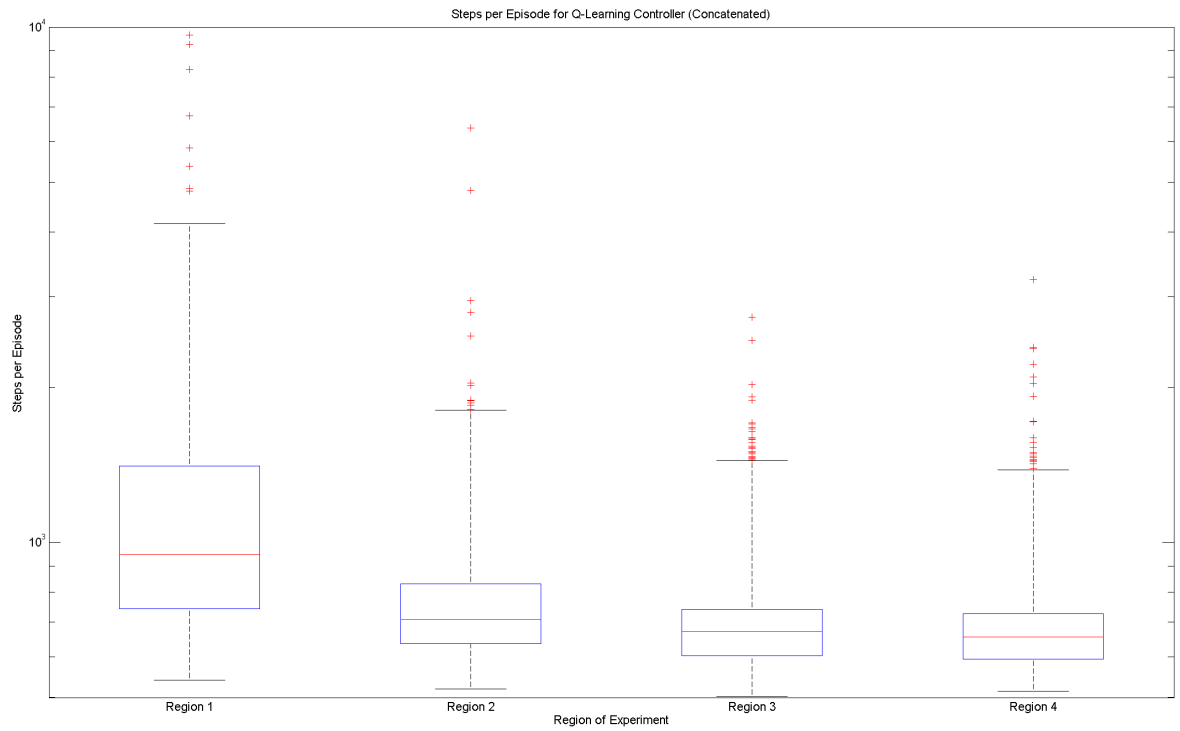


Figure 6.25: Distribution of concatenated episode lengths for QL controller ( $\lambda = 0.9$ ).

<b>Overall Performance</b>				
Total Length (Episodes)	6980			
Steady State Median Length (Steps)	661			
	<b>Region 1</b>	<b>Region 2</b>	<b>Region 3</b>	<b>Region 4</b>
Length (Episodes)	157	483	3170	3170
<b>Episode Length (Steps)</b>				
Mean	1588	785	693	674
Median	947	709	672	655
Standard Deviation	2411	299	129	115
Lower Quartile	744	636	602	655
Upper Quartile	1407	832	743	727
Minimum	541	520	502	514
Maximum	22516	6362	2738	3238

Table 6.14: KPIs for QL controller ( $\lambda = 0.9$ ).

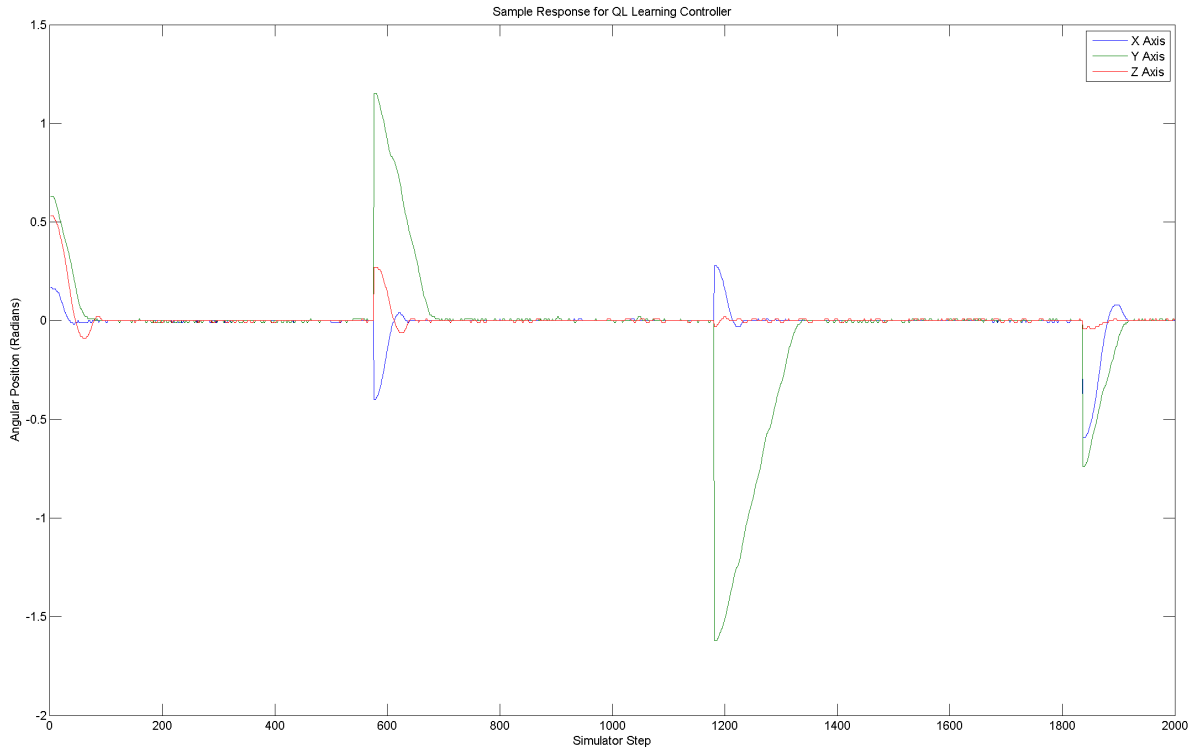


Figure 6.26: Sample time-domain response for QL controller ( $\lambda = 0.9$ ).

Parameter	Value
$\epsilon$	0.05
$\alpha$	0.2
$\gamma$	0.9
$\lambda$	0.95
$\sigma_\theta$	128
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	{0.2, 1.0, 0.2}

Table 6.15: Tunable parameters for QL controller ( $\lambda = 0.9$ ).

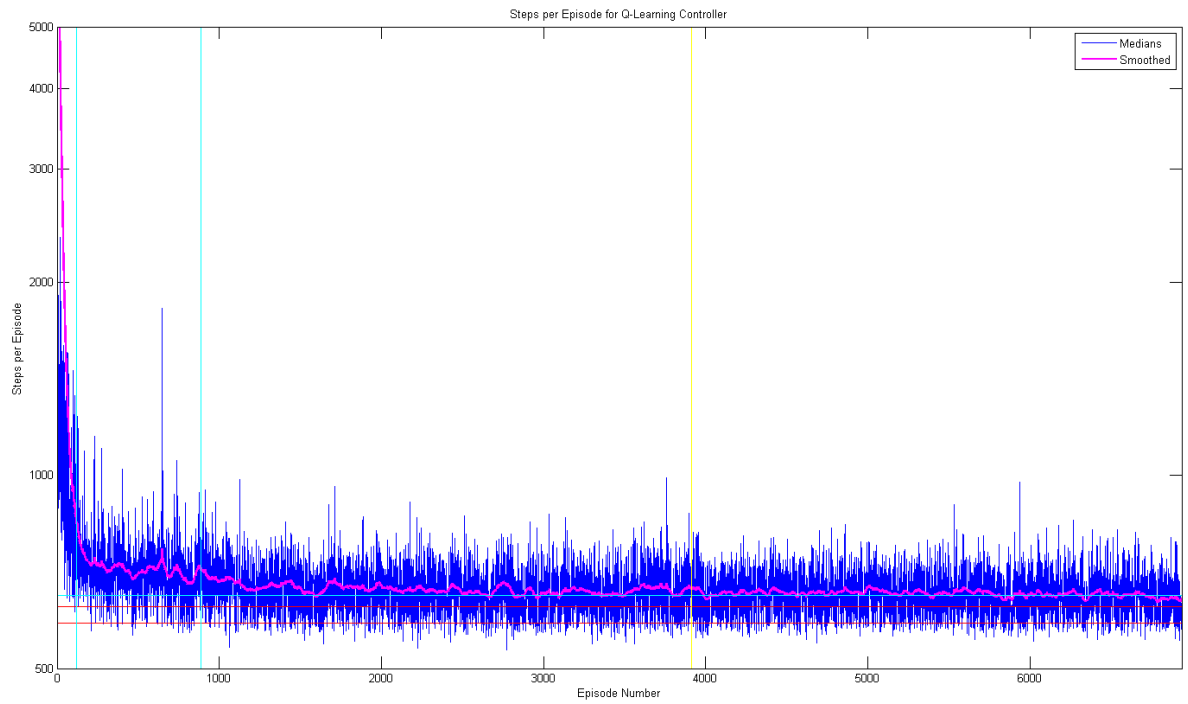


Figure 6.27: Smoothed episode lengths for QL controller ( $\lambda = 0.95$ ).

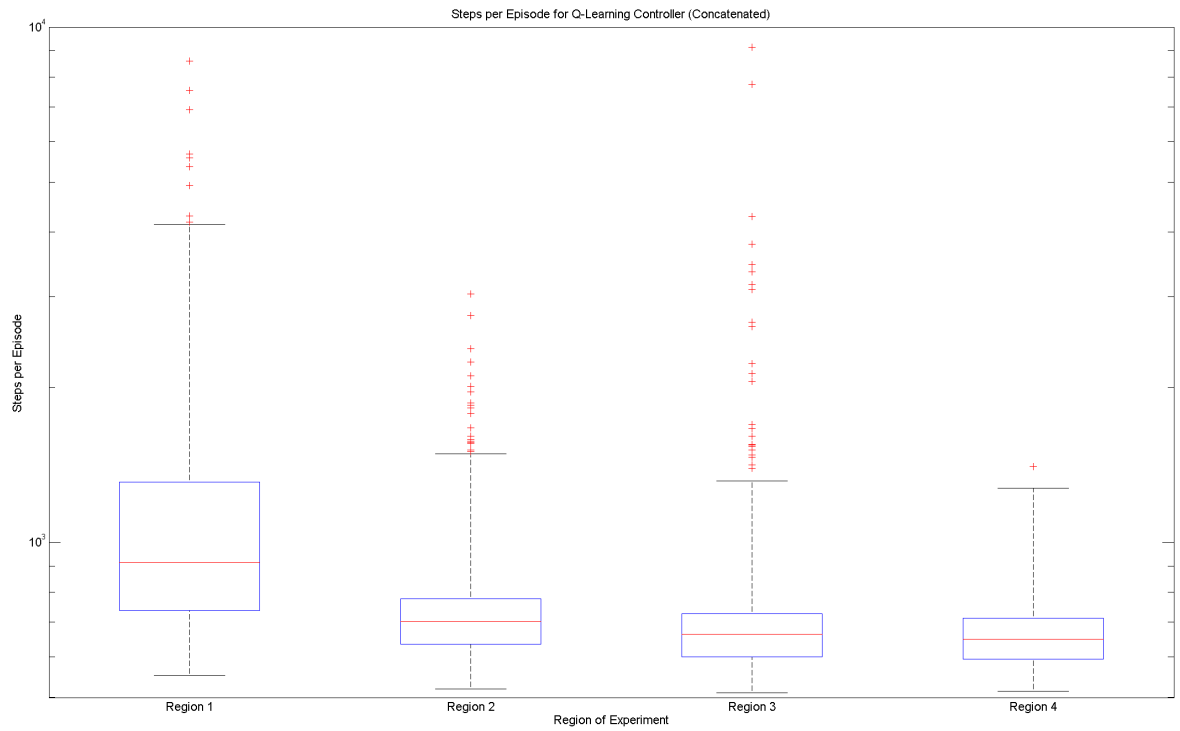


Figure 6.28: Distribution of concatenated episode lengths for QL controller ( $\lambda = 0.95$ ).



<b>Overall Performance</b>				
Total Length (Episodes)	6935			
Steady State Median Length (Steps)	651			
	<b>Region 1</b>	<b>Region 2</b>	<b>Region 3</b>	<b>Region 4</b>
Length (Episodes)	122	764	3025	3024
<b>Episode Length (Steps)</b>				
Mean	1955	837	679	662
Median	915	702	663	649
Standard Deviation	7726	2593	172	86
Lower Quartile	738	635	599	593
Upper Quartile	1308	778	727	713
Minimum	551	519	511	514
Maximum	127830	85029	9134	1404

Table 6.16: KPIs for QL controller ( $\lambda = 0.95$ ).

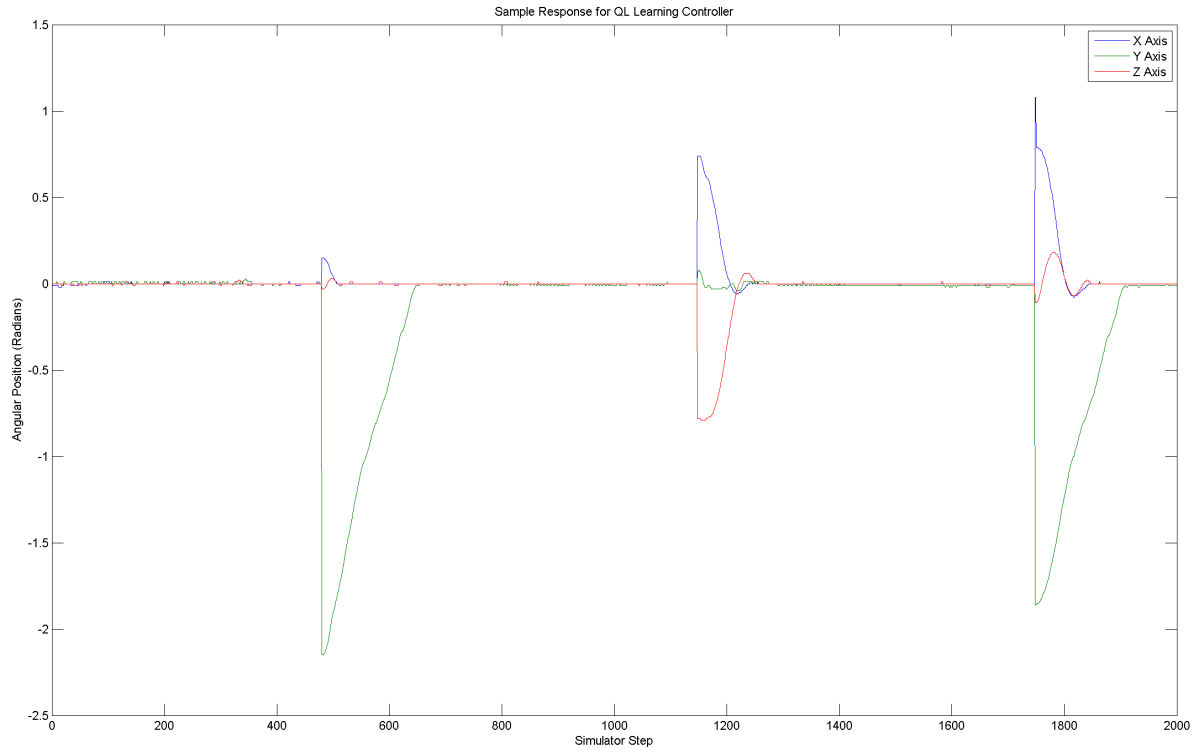


Figure 6.29: Sample time-domain response for QL controller ( $\lambda = 0.95$ ).

Parameter	Value
$\epsilon$	0.05
$\alpha$	0.1
$\gamma$	0.9
$\lambda$	0.7
$\sigma_\theta$	128
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	{0.2, 1.0, 0.2}

Table 6.17: Tunable parameters for QL controller ( $\lambda = 0.7$ ).

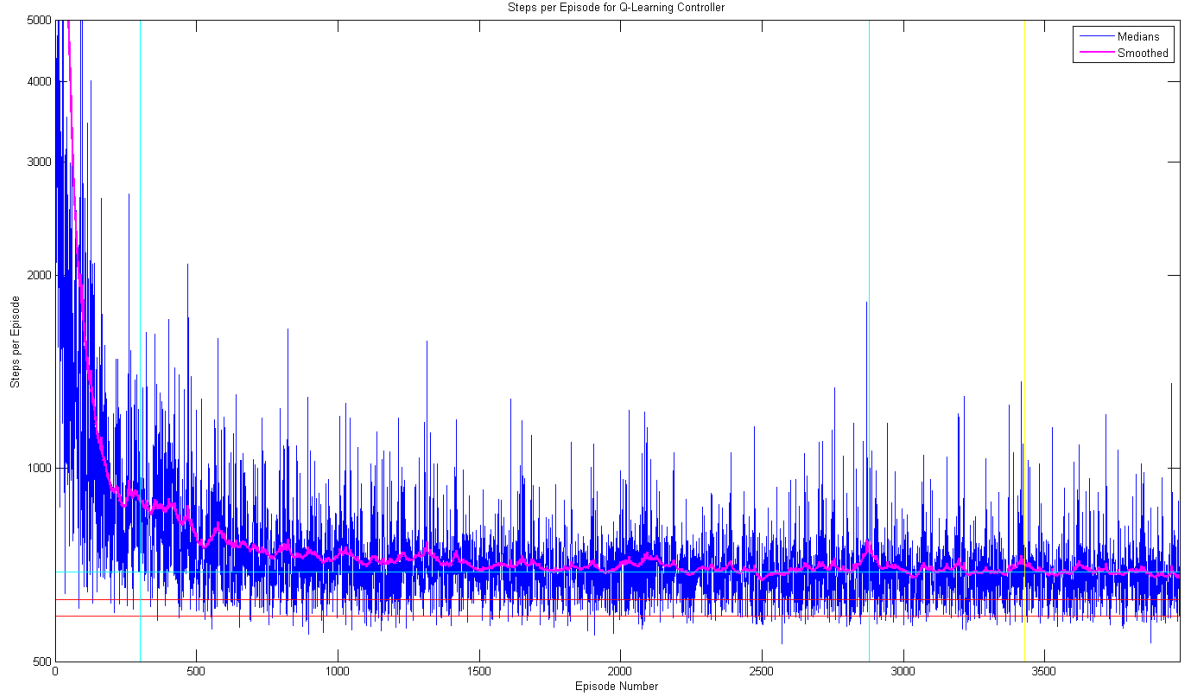


Figure 6.30: Smoothed episode lengths for QL controller ( $\lambda = 0.7$ ).

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.31. The key performance characteristics derived from the experiment are also shown in Table 6.18.

Comparison of these key performance values with those of the initial baseline experiment suggests that the reduction in  $\lambda$  value results in significantly degraded performance. Whilst there is a moderate increase in median episode length, the large reduction in the total number of completed episodes is due to instability in the performance of the controller; the standard deviation of episode length remains high throughout the experiment, due to the presence of many more long outlier episodes.

Finally, consideration is also given to the time-domain response of the controller. A sample plot of the response is shown in Figure 6.32. The response of the controller remains similar to

<b>Overall Performance</b>				
Total Length (Episodes)	3977			
Steady State Median Length (Steps)	690			
	<b>Region 1</b>	<b>Region 2</b>	<b>Region 3</b>	<b>Region 4</b>
Length (Episodes)	300	2580	549	548
<b>Episode Length (Steps)</b>				
Mean	1989	890	840	815
Median	1021	698	679	675
Standard Deviation	4171	1059	750	1250
Lower Quartile	735	618	611	606
Upper Quartile	1852	821	760	748
Minimum	534	511	526	518
Maximum	69819	49819	14123	44198

Table 6.18: KPIs for QL controller ( $\lambda = 0.7$ ).

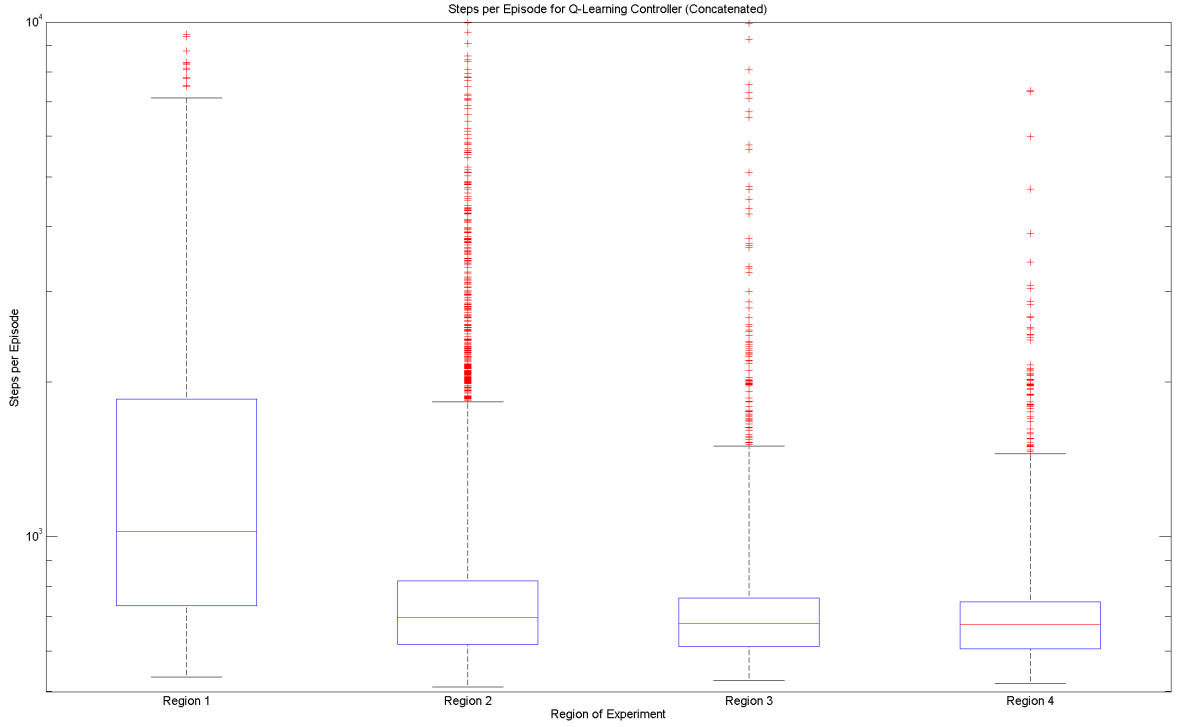


Figure 6.31: Distribution of concatenated episode lengths for QL controller ( $\lambda = 0.7$ ).

that of the initial baseline, with underdamping present in the pitch and roll axes, and severe overdamping in the yaw axis for large errors.

The experiments indicate that whilst there is little performance gain to be had in selecting a value for  $\lambda$  within the range  $0.8 - 0.95$ , there is considerable performance penalty for selecting too low a value; the performance of the controller becomes very unstable. Accordingly, best practice would seem to simply select a value which will be comfortably within the region known to deliver stable performance.

Since there appears to be minimal performance gain in selecting a value above the original value of  $\lambda = 0.8$ , and because this value has worked adequately in experiments thus far, further experiments will continue to use the original value.

### 6.2.5 Varying Velocity Reward Gain $K_{\dot{\theta}}$

A set of experiments were then conducted to characterise the effect which variations in the reward scheme parameter  $K_{\dot{\theta}}$  would have on the performance of the controller. In each experiment, this parameter was adjusted, whilst the remainder of the parameters remained fixed to values used previously, to allow comparison with previous experiments.

**Flattened Gains ( $K_{\dot{\theta}} = 0.2$ )** In previous experiments, individual values for  $K_{\dot{\theta}}$  were used for each degree of freedom of the vehicle. As mentioned in §6.2.1, this configuration was selected because the Y (yaw) axis has a larger range of motion than the X & Z axes. However, in the time-domain response of previous experiments, the Y-axis response has typically been significantly

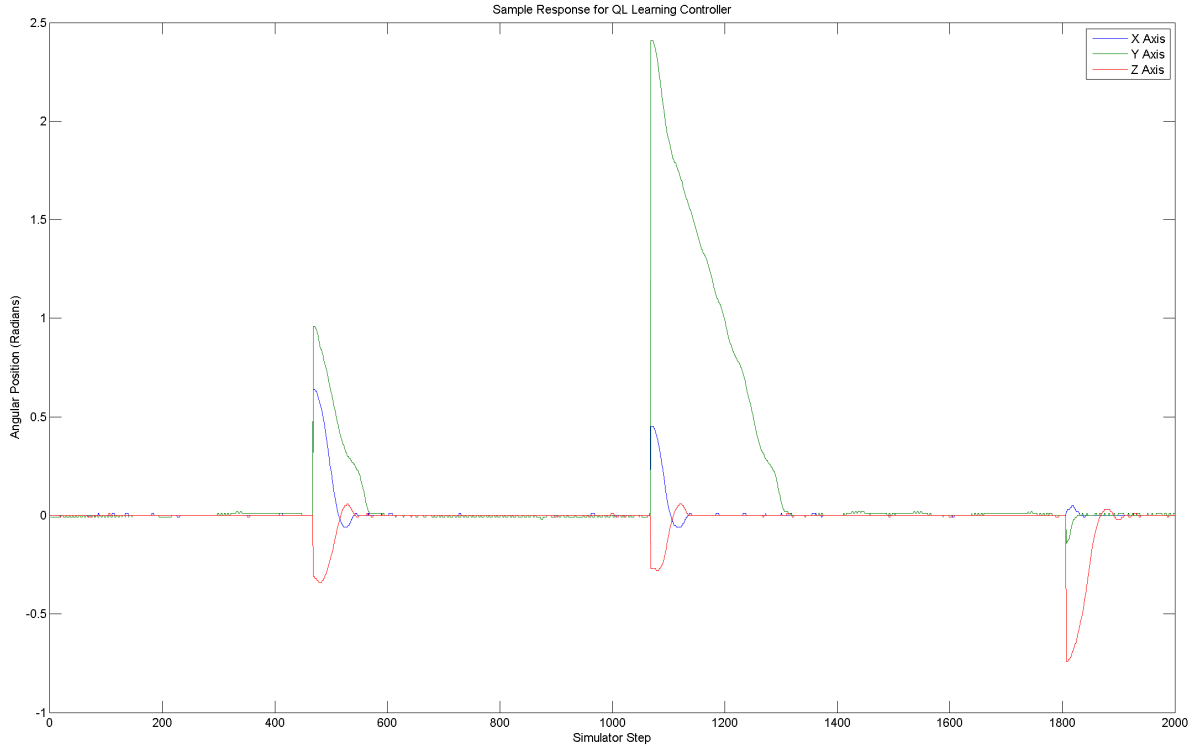


Figure 6.32: Sample time-domain response for QL controller ( $\lambda = 0.7$ ).

overdamped, whilst the X & Z axes have been underdamped. This suggests that perhaps the significant difference in velocity reward gain for the Y-axis was a poor choice. To confirm this, an experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.19; parameters are kept the same as for the initial baseline experiment, but for  $K_{\dot{\theta}}$ , which is set to 0.2 for all three axes. The resulting smoothed set of episode lengths is shown in Figure 6.33.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.34. The key performance characteristics derived from the experiment are also shown in Table 6.20.

Comparison of these key performance values with those of the initial baseline experiment suggests that flattening the  $K_{\dot{\theta}}$  values across all axes results in a significant improvement in performance, throughout the experiment. Steady-state median episode length is reduced by around 10% from that of the initial baseline, with a corresponding increase in the total number of episodes

Parameter	Value
$\epsilon$	0.05
$\alpha$	0.5
$\gamma$	0.9
$\lambda$	0.8
$\sigma_{\theta}$	128
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	0.2

Table 6.19: Tunable parameters for QL controller ( $K_{\dot{\theta}} = 0.2$ ).

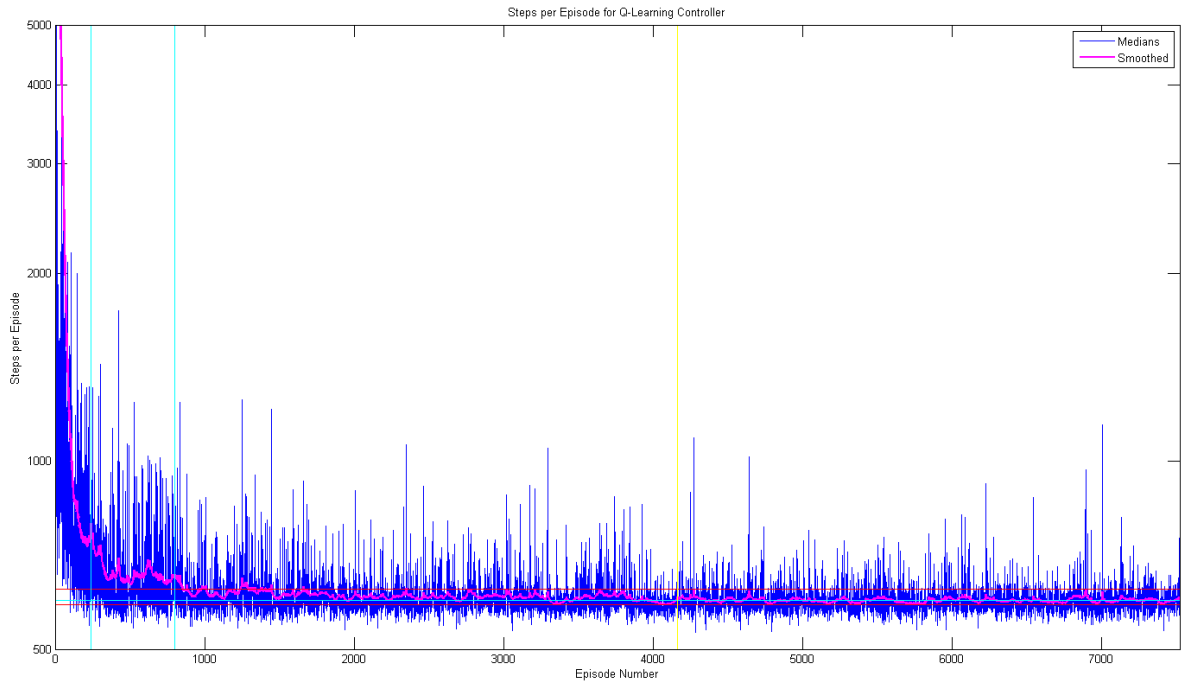


Figure 6.33: Smoothed episode lengths for QL controller ( $K_{\dot{\theta}} = 0.2$ ).

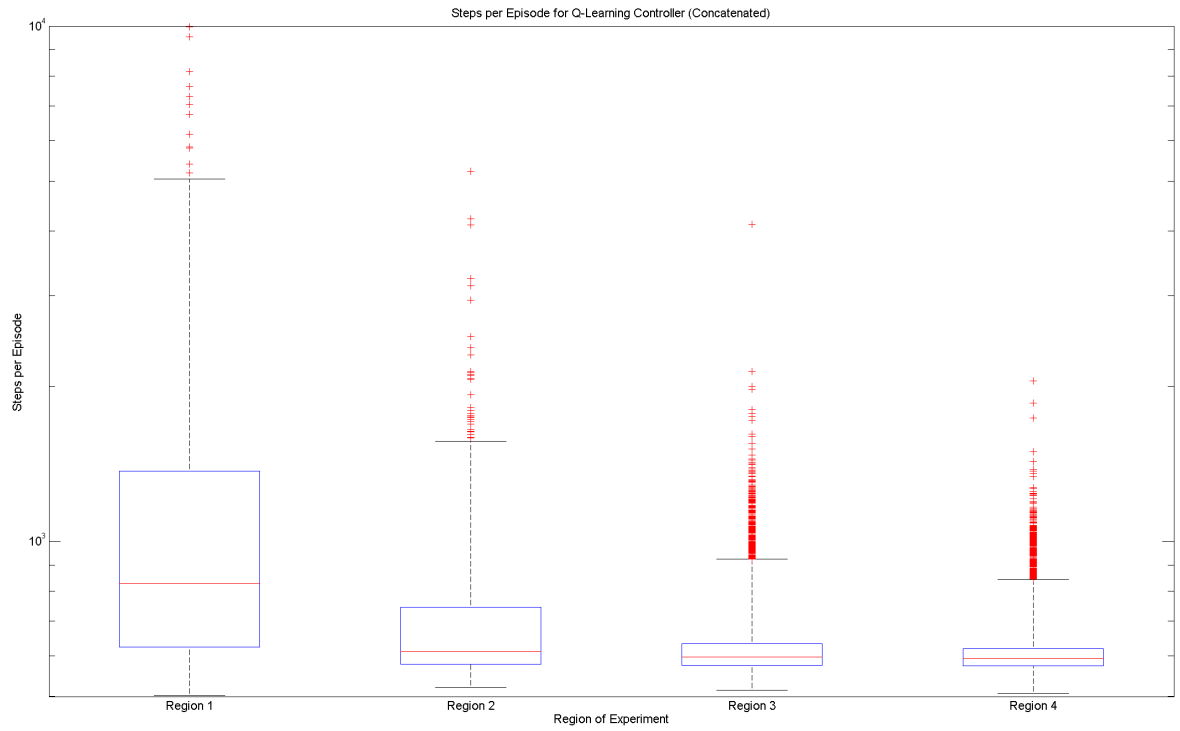


Figure 6.34: Distribution of concatenated episode lengths for QL controller ( $K_{\dot{\theta}} = 0.2$ ).

<b>Overall Performance</b>				
Total Length (Episodes)	7522			
Steady State Median Length (Steps)	598			
	Region 1	Region 2	Region 3	Region 4
Length (Episodes)	240	563	3360	3359
<b>Episode Length (Steps)</b>				
Mean	1518	742	630	613
Median	828	611	596	592
Standard Deviation	3483	594	117	83
Lower Quartile	625	579	575	574
Upper Quartile	1371	745	633	619
Minimum	502	521	515	507
Maximum	50113	15891	4118	2047

Table 6.20: KPIs for QL controller ( $K_{\dot{\theta}} = 0.2$ ).

completed. The steady-state median episode length is now only marginally higher than for the PID controller. The distribution of episode lengths is also tightened, with standard deviation reduced throughout the experiment. There is a slight increase in the settling time of the learner, but not sufficient to be significant (this is expected, since the reward scheme only affect *what* behaviour the controller learns, not how it learns).

Finally, consideration is also given to the time-domain response of the controller. A sample plot of the response is shown in Figure 6.35. The response of the controller now appears consistent across all three axes; underdamped for large errors, but oscillations are arrested fairly quickly.

**High Gain ( $K_{\dot{\theta}} = 0.35$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.21; the value of  $K_{\dot{\theta}}$  (the velocity reward gain) is increased to 0.35 (across all three axes). The resulting smoothed set of episode lengths is shown in Figure 6.36.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.37. The key performance characteristics derived from the experiment are also shown in Table 6.22.

Comparison of these key performance values with those of the previous experiment suggests that the increase in  $K_{\dot{\theta}}$  value results in further improved performance throughout the experiment. The steady-state median episode length is further reduced, such that it is now directly competitive with that of the conventional PID controller. The standard deviation of episode lengths is also reduced throughout the experiment, although still at least double that of the PID controller. As expected, there is no significant changes to the learning rate of the controller.

Finally, consideration is also given to the time-domain response of the controller. A sample

Parameter	Value
$\epsilon$	0.05
$\alpha$	0.5
$\gamma$	0.9
$\lambda$	0.8
$\sigma_{\theta}$	128
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	0.35

Table 6.21: Tunable parameters for QL controller ( $K_{\dot{\theta}} = 0.35$ ).

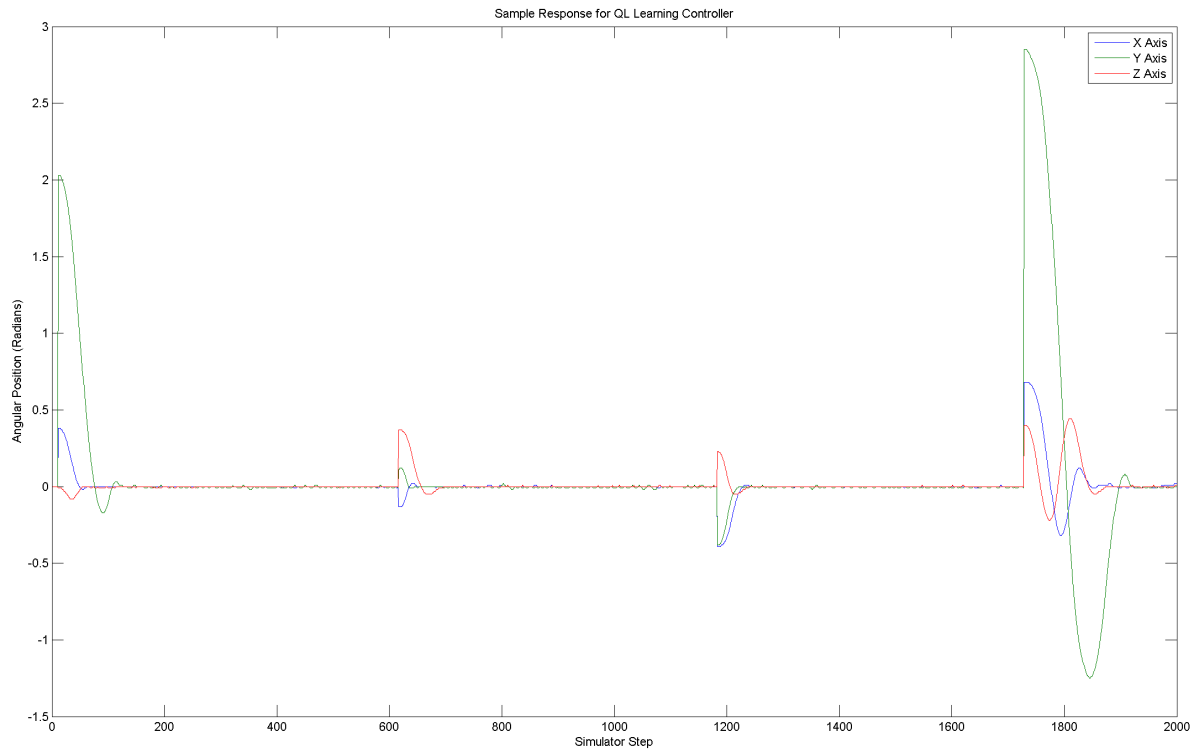


Figure 6.35: Sample time-domain response for QL controller ( $K_{\dot{\theta}} = 0.2$ ).

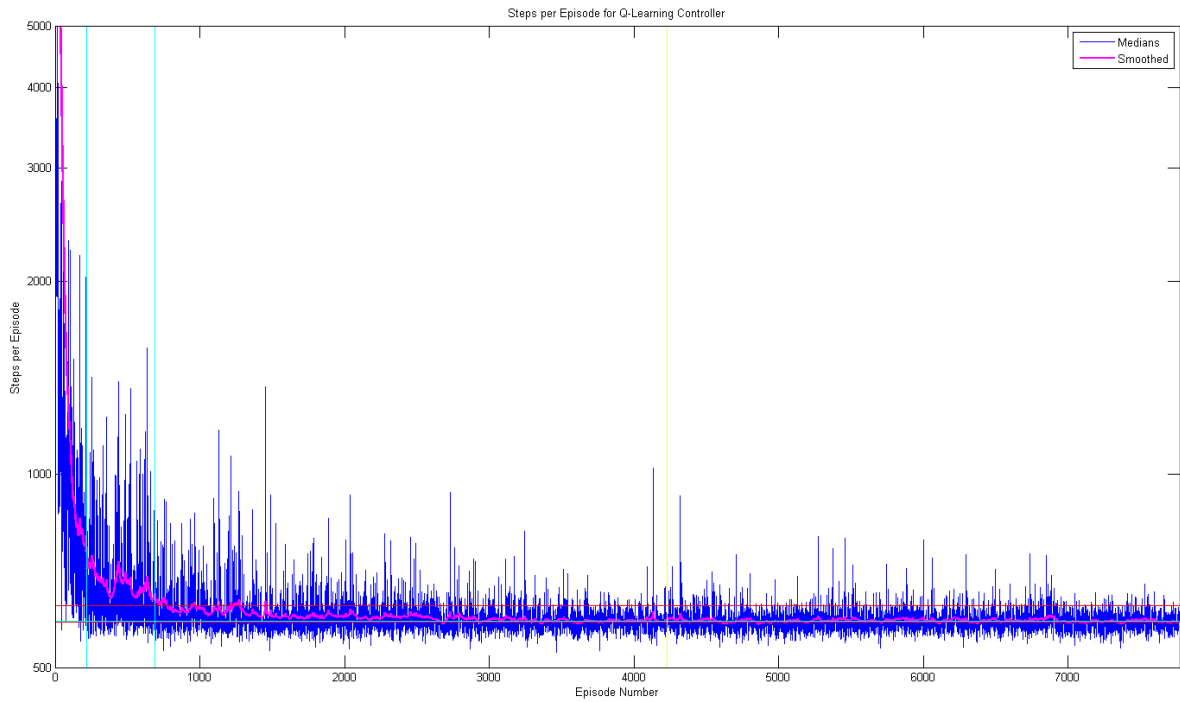


Figure 6.36: Smoothed episode lengths for QL controller ( $K_{\dot{\theta}} = 0.35$ ).

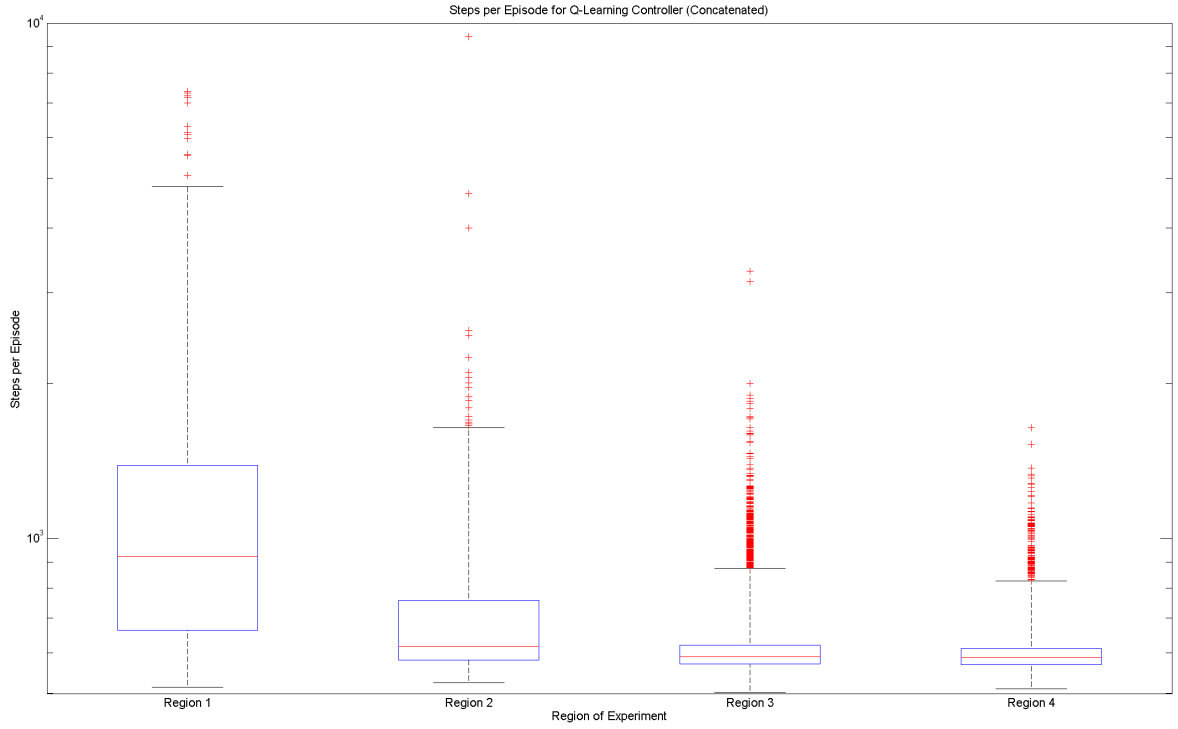


Figure 6.37: Distribution of concatenated episode lengths for QL controller ( $K_{\dot{\theta}} = 0.35$ ).

<b>Overall Performance</b>				
Total Length (Episodes)	7771			
Steady State Median Length (Steps)	590			
	Region 1	Region 2	Region 3	Region 4
Length (Episodes)	215	475	3541	3540
<b>Episode Length (Steps)</b>				
Mean	1472	733	619	600
Median	922	618	590	588
Standard Deviation	2921	363	110	62
Lower Quartile	662	581	570	569
Upper Quartile	1385	762	621	612
Minimum	514	525	502	511
Maximum	42333	9411	3299	1641

Table 6.22: KPIs for QL controller ( $K_{\dot{\theta}} = 0.35$ ).



plot of the response is shown in Figure 6.38. The response of the the X & Z axes remains underdamped, similarly to the previous experiment, but the Y (yaw) axis response displays some of the overdamped characteristics present in the initial baseline experiment.

**High Gain ( $K_{\dot{\theta}} = 0.5$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.23; the value of  $K_{\dot{\theta}}$  (the velocity reward gain) is increased to 0.5. The resulting smoothed set of episode lengths is shown in Figure 6.39.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.40. The key performance characteristics derived from the experiment are also shown in Table 6.24.

Comparison of these key performance values against those of the two previous experiments suggests that the further increase in  $K_{\dot{\theta}}$  value results in a slight reduction in control performance against the  $K_{\dot{\theta}} = 0.2$  case and a more pronounced reduction against the  $K_{\dot{\theta}} = 0.35$  case. There are slight increases in both episode length median and standard deviation throughout all regions of the experiment. This suggests that there will be some maxima value for  $K_{\dot{\theta}}$  (between 0.2 and 0.5 in this case). As before, there is no significant affect in terms of learning performance.

Finally, consideration is also given to the time-domain response of the controller. A sample plot of the response is shown in Figure 6.41. The response of the X & Z axes remains underdamped, similarly to the previous experiment, but the Y (yaw) axis response displays some of the overdamped characteristics present in the initial baseline experiment.

**Low Gain ( $K_{\dot{\theta}} = 0.1$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.25; the value of  $K_{\dot{\theta}}$  (the velocity reward gain) is reduced to 0.1. The resulting smoothed set of episode lengths is shown in Figure 6.42.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.43. The key performance characteristics derived from the experiment are also shown in Table 6.26.

Comparison of these key performance values with those of the  $K_{\dot{\theta}} = 0.2$  experiment suggests that the reduction in  $K_{\dot{\theta}}$  value results in a significant reduction in control performance. The median episode length is significantly longer than for the  $K_{\dot{\theta}} = 0.2$  case, although still an improvement upon the initial baseline experiment. Standard deviation of episode lengths is greater than that of the initial baseline. The greater instability in the performance results in the learner taking longer to settle, which increases the lengths of regions one and two.

Finally, consideration is also given to the time-domain response of the controller. A sample plot of the response is shown in Figure 6.44. The response of all three axes are underdamped; there

Parameter	Value
$\epsilon$	0.05
$\alpha$	0.5
$\gamma$	0.9
$\lambda$	0.8
$\sigma_{\theta}$	128
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	0.5

Table 6.23: Tunable parameters for QL controller ( $K_{\dot{\theta}} = 0.5$ ).

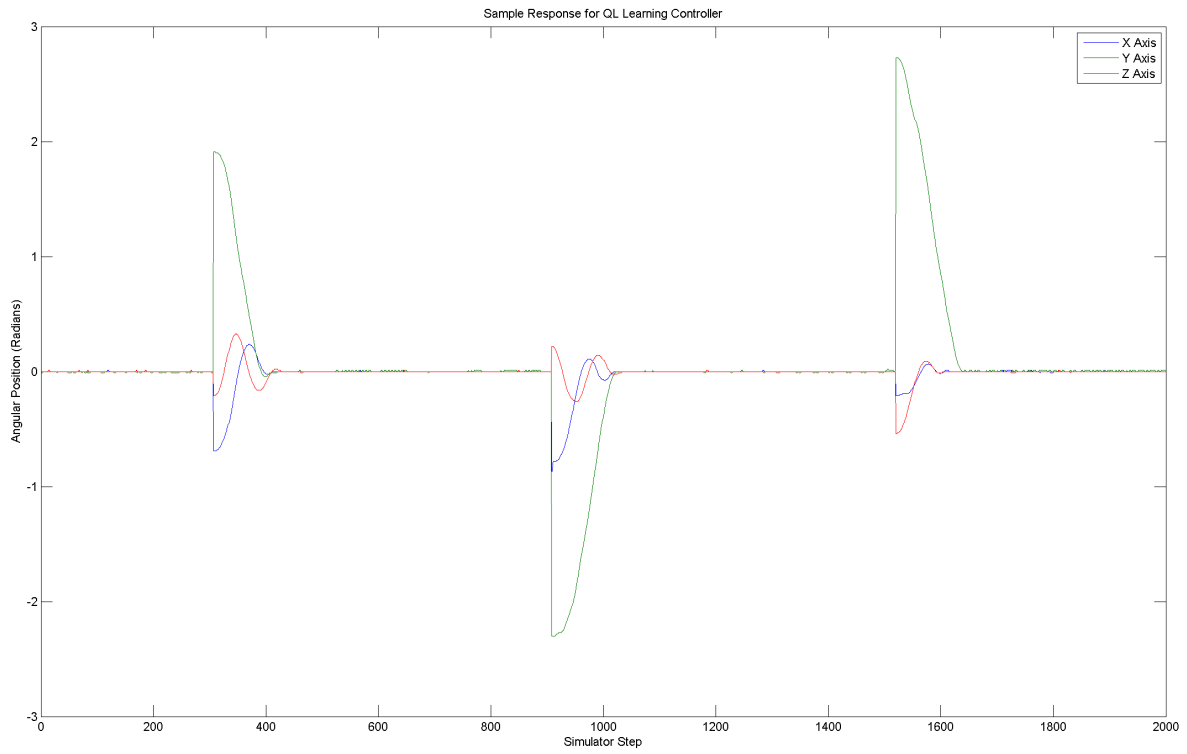


Figure 6.38: Sample time-domain response for QL controller ( $K_{\dot{\theta}} = 0.35$ ).

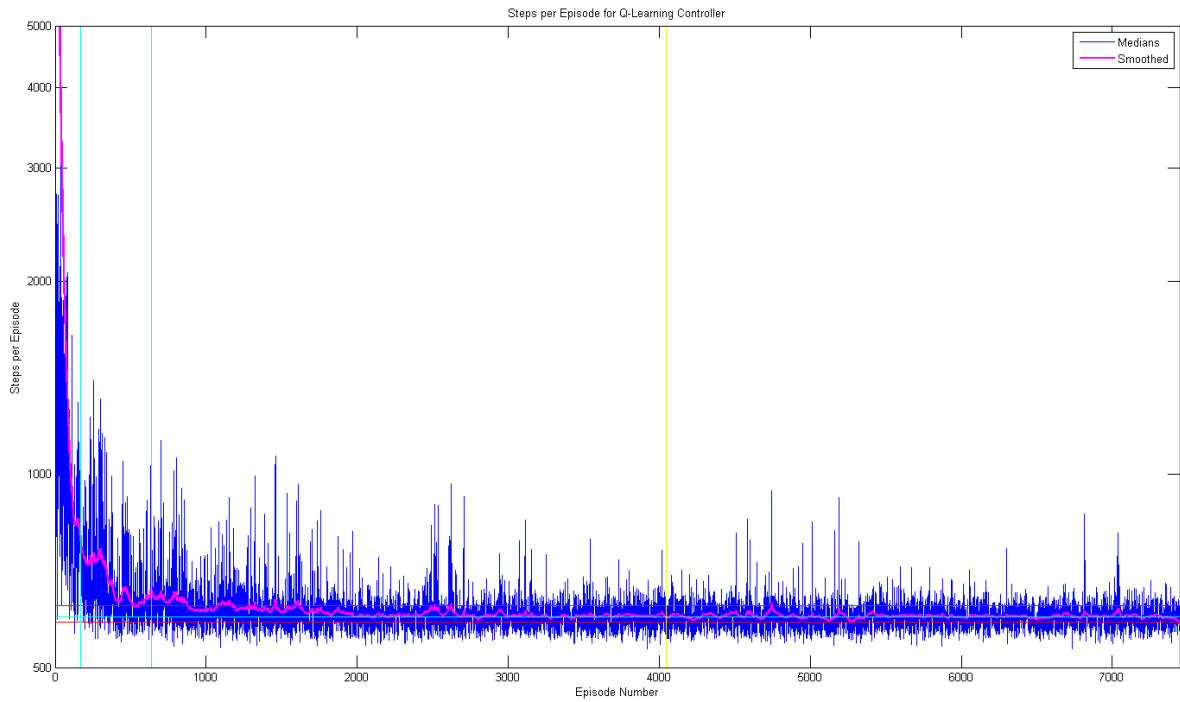


Figure 6.39: Smoothed episode lengths for QL controller ( $K_{\dot{\theta}} = 0.5$ ).

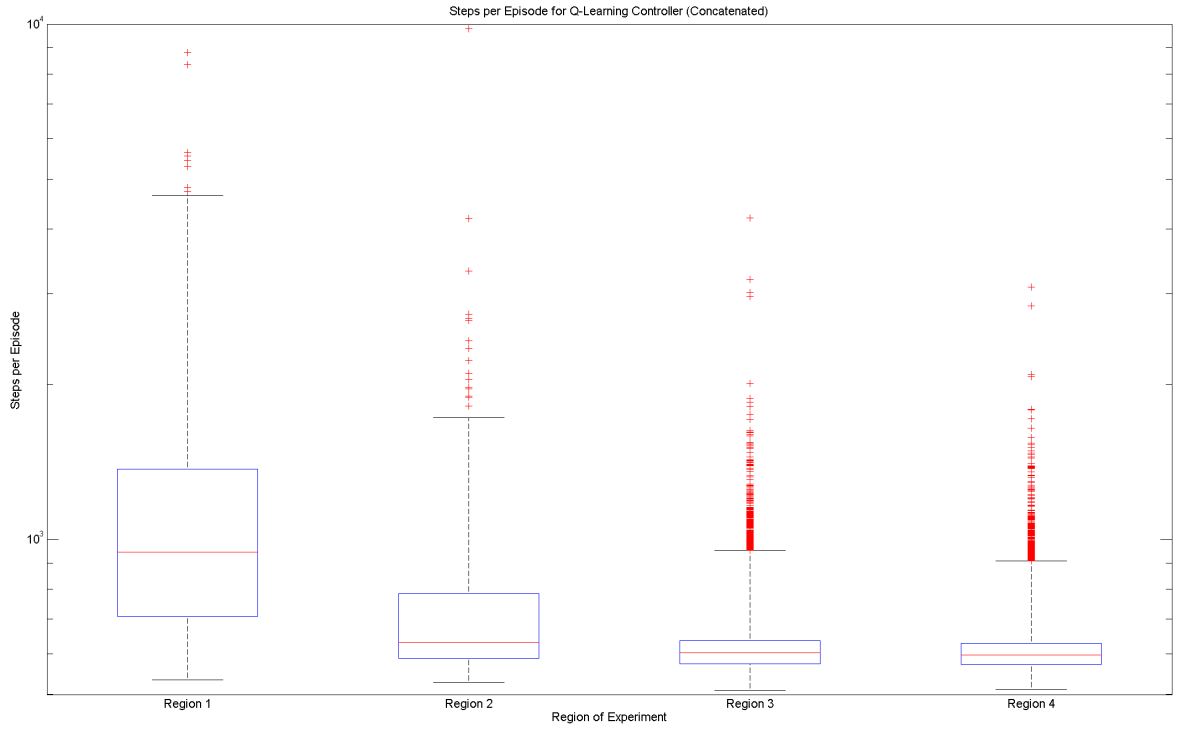


Figure 6.40: Distribution of concatenated episode lengths for QL controller ( $K_{\dot{\theta}} = 0.5$ ).

<b>Overall Performance</b>				
Total Length (Episodes)	7447			
Steady State Median Length (Steps)	600			
	Region 1	Region 2	Region 3	Region 4
Length (Episodes)	170	470	3404	3404
<b>Episode Length (Steps)</b>				
Mean	1493	757	631	617
Median	945	632	602	597
Standard Deviation	2838	778	126	99
Lower Quartile	710	588	573	572
Upper Quartile	1370	785	636	628
Minimum	534	527	509	512
Maximum	38585	26569	4208	3084

Table 6.24: KPIs for QL controller ( $K_{\dot{\theta}} = 0.5$ ).

Parameter	Value
$\epsilon$	0.05
$\alpha$	0.5
$\gamma$	0.9
$\lambda$	0.8
$\sigma_{\theta}$	128
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	0.1

Table 6.25: Tunable parameters for QL controller ( $K_{\dot{\theta}} = 0.1$ ).

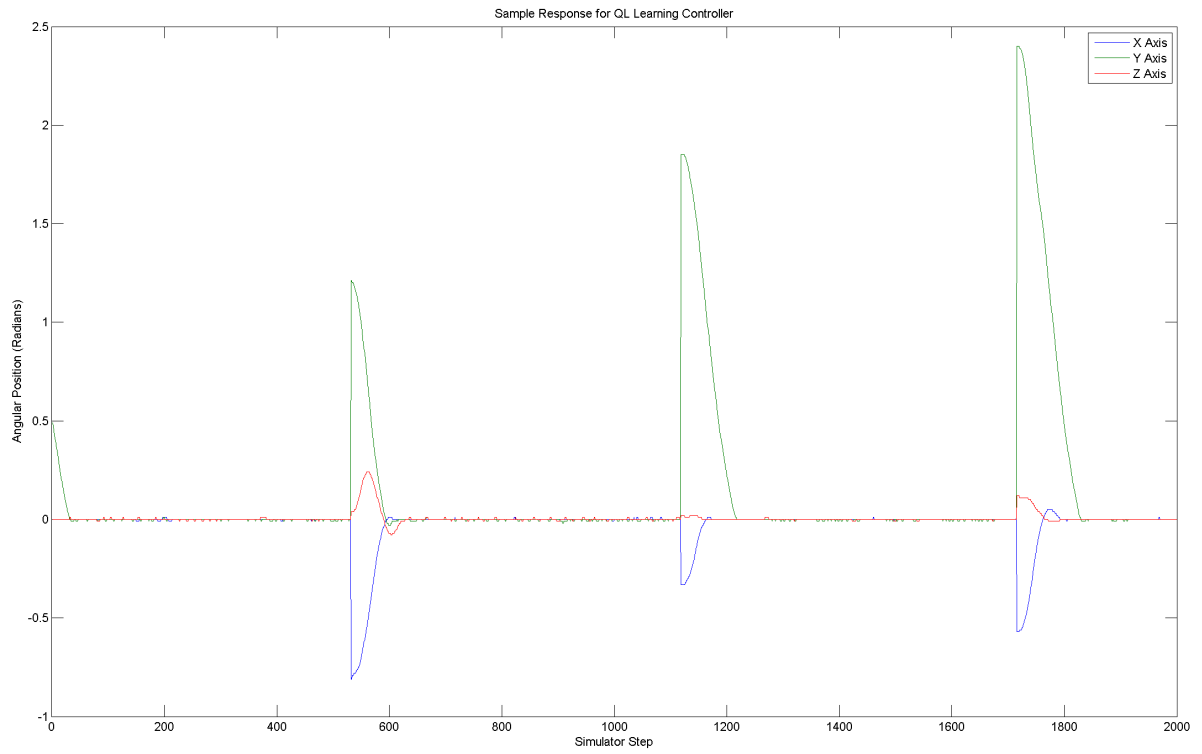


Figure 6.41: Sample time-domain response for QL controller ( $K_{\dot{\theta}} = 0.5$ ).

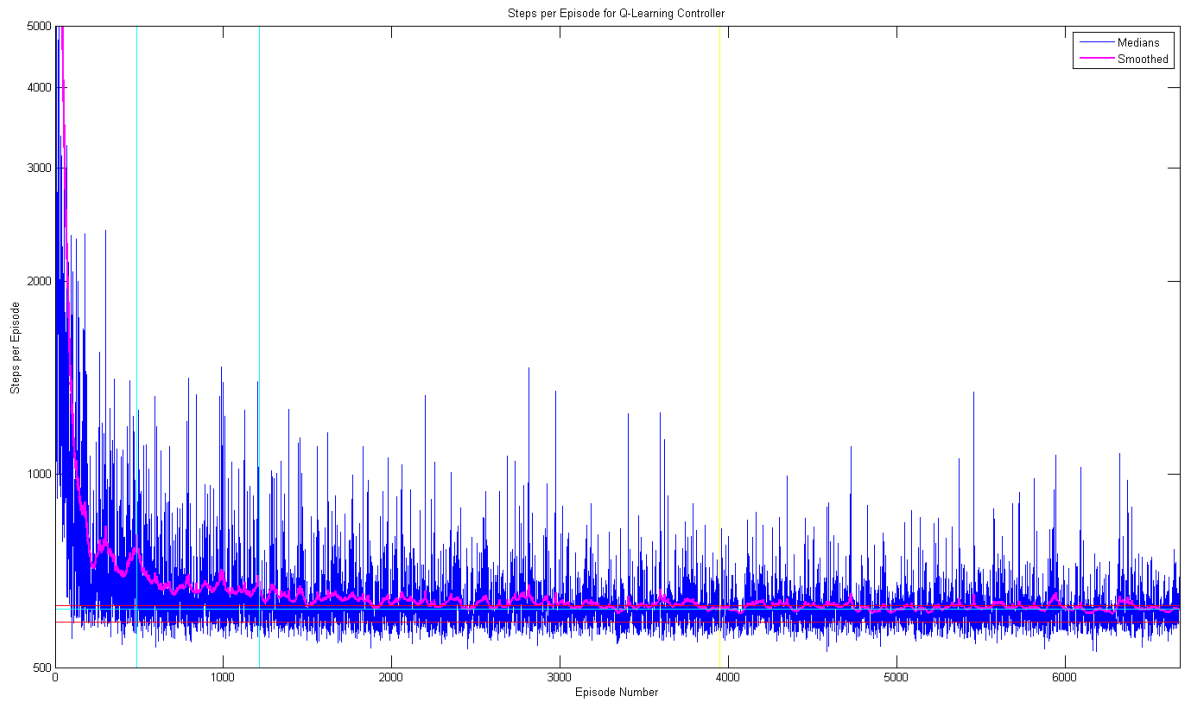


Figure 6.42: Smoothed episode lengths for QL controller ( $K_{\dot{\theta}} = 0.1$ ).

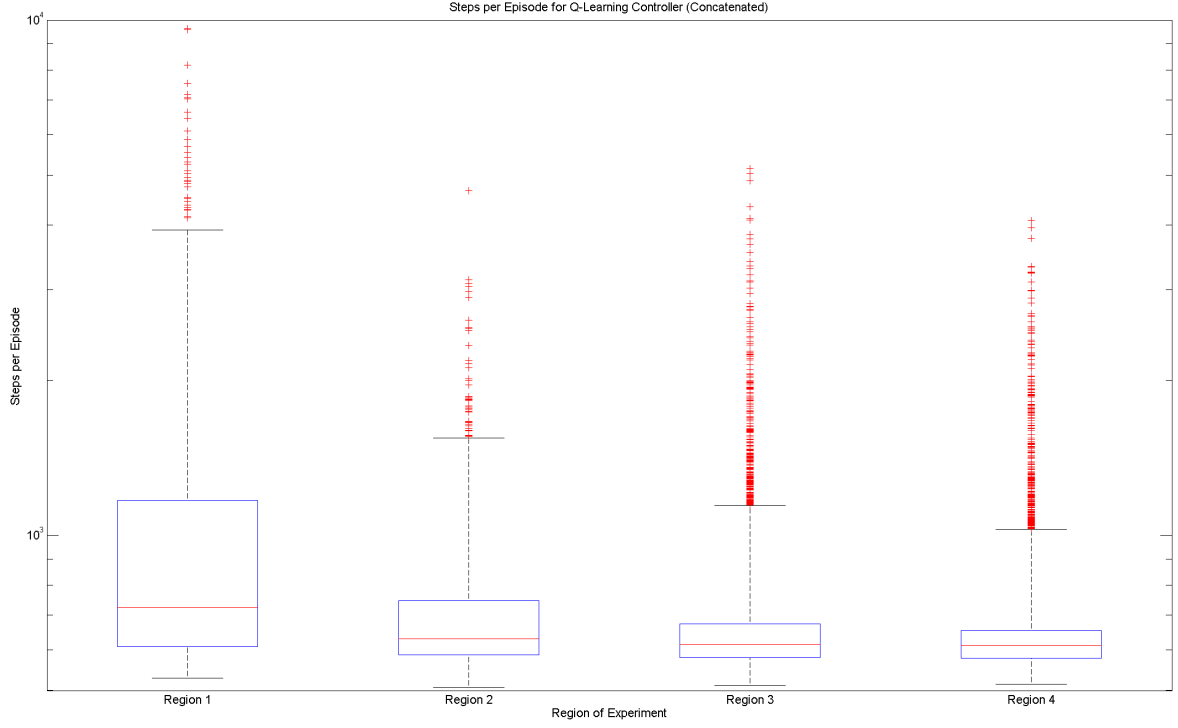


Figure 6.43: Distribution of concatenated episode lengths for QL controller ( $K_{\dot{\theta}} = 0.1$ ).

<b>Overall Performance</b>				
Total Length (Episodes)	6682			
Steady State Median Length (Steps)	617			
	Region 1	Region 2	Region 3	Region 4
Length (Episodes)	484	730	2734	2734
<b>Episode Length (Steps)</b>				
Mean	1371	731	678	657
Median	726	630	615	611
Standard Deviation	2960	290	275	206
Lower Quartile	608	587	579	578
Upper Quartile	1168	747	673	653
Minimum	529	507	512	514
Maximum	53770	4662	11637	4081

Table 6.26: KPIs for QL controller ( $K_{\dot{\theta}} = 0.1$ ).

is substantial initial overshoot in the step response, though the response decays immediately following this.

**Zero Gain ( $K_{\dot{\theta}} = 0.0$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.27; the value of  $K_{\dot{\theta}}$  (the velocity reward gain) is reduced to 0.0. This makes the reward function used by the controller solely a function of the angular position of the vehicle. The resulting smoothed set of episode lengths is shown in Figure 6.45.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.46. The key performance characteristics derived from the experiment are also shown in Table 6.28.

Comparison of these key performance values with those of the  $K_{\dot{\theta}} = 0.2$  and  $K_{\dot{\theta}} = 0.1$  experiments suggests that complete reduction of  $K_{\dot{\theta}}$  value results in further degraded control performance. The median episode length is further increased throughout the experiment (though still better than that of the initial baseline). The standard deviation of episode lengths is consistently greater than in the  $K_{\dot{\theta}} = 0.2$  case, though settles to better than those of the  $K_{\dot{\theta}} = 0.1$  case (which is probably indicative of the small sample sizes involved in the project more than anything else).

Finally, consideration is also given to the time-domain response of the controller. A sample plot of the response is shown in Figure 6.47. The response of all three axes are underdamped; there is substantial initial overshoot in the step response, though the oscillation decays immediately following this.

The experiments show that there are substantial improvements in performance to be had by correctly selecting values for the reward scheme, and that performance decreases significantly for sub-optimal values. This is understandable: the controller is only able to operate as to optimize the observed reward; if the chosen reward scheme does not accurately represent desired behaviour of the environment, then the controller will not deliver optimal behaviour.

Since the value of  $K_{\dot{\theta}} = 0.35$  (across all three axes) appears to offer an improvement in performance, without any substantial penalties, compared to the original value of  $K_{\dot{\theta}} = 0.2$ , further experiments will use the higher value in order to best demonstrate to potential of the controller.

### 6.2.6 Varying Displacement ARBFN Resolution $\sigma_{\theta}$

A set of experiments were then conducted to characterise the effect which variations in angular displacement resolution parameter  $\sigma_{\theta}$  for the Q-Function estimator would have on the performance

Parameter	Value
$\epsilon$	0.05
$\alpha$	0.5
$\gamma$	0.9
$\lambda$	0.8
$\sigma_{\theta}$	128
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	0.0

Table 6.27: Tunable parameters for QL controller ( $K_{\dot{\theta}} = 0.0$ ).

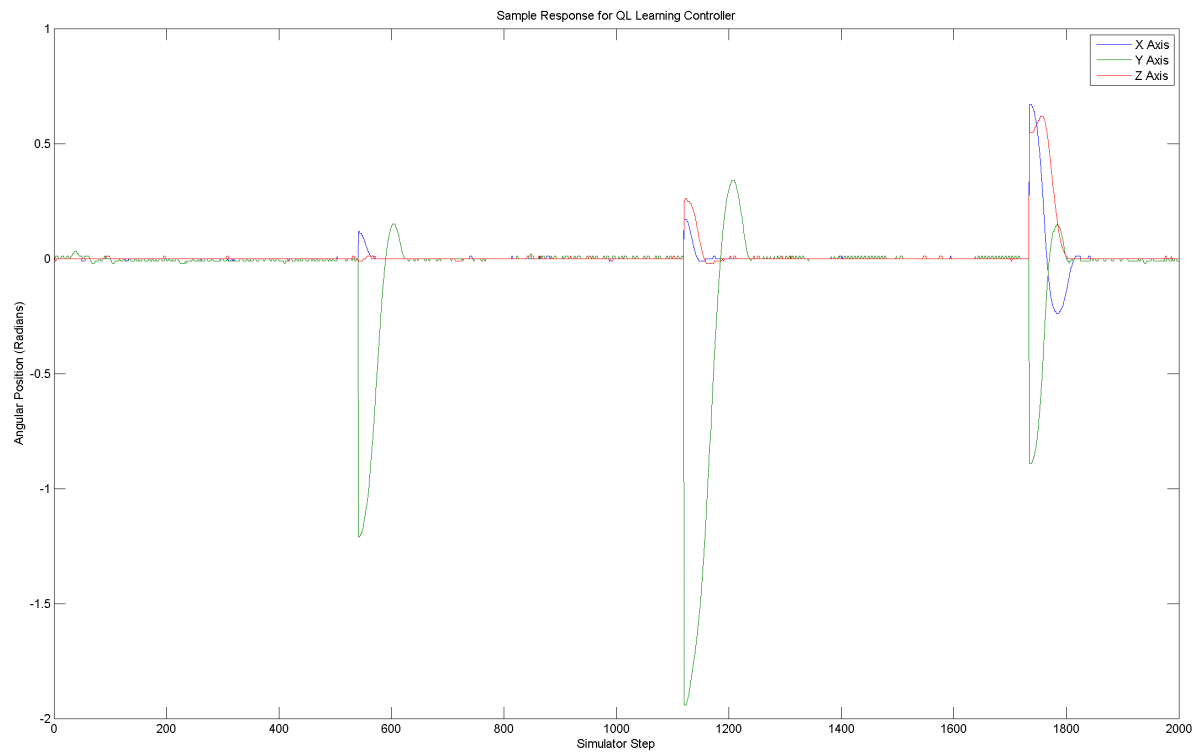


Figure 6.44: Sample time-domain response for QL controller ( $K_{\dot{\theta}} = 0.1$ ).

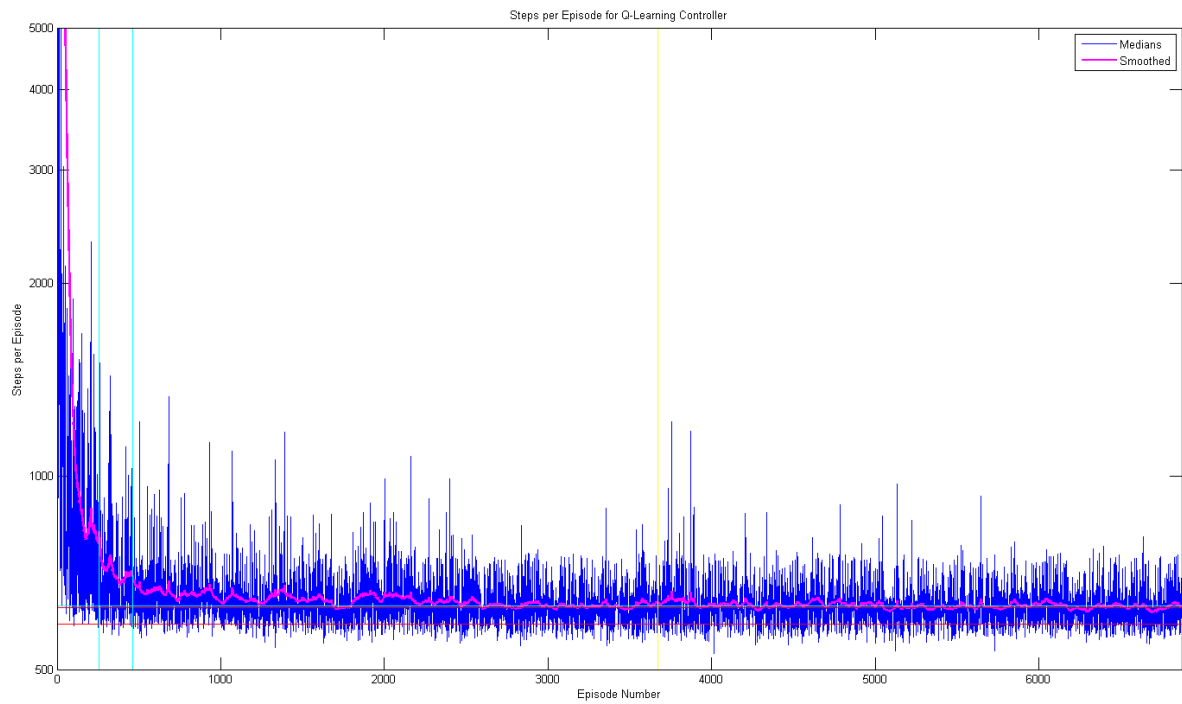


Figure 6.45: Smoothed episode lengths for QL controller ( $K_{\dot{\theta}} = 0.0$ ).

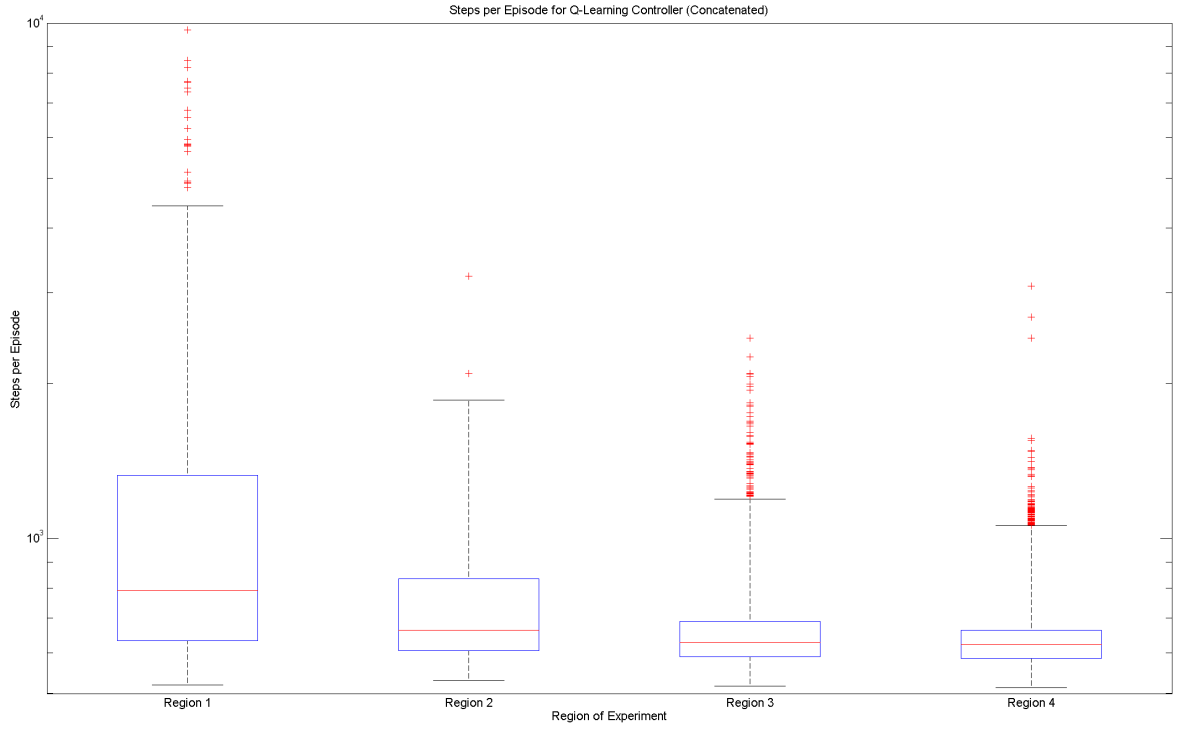


Figure 6.46: Distribution of concatenated episode lengths for QL controller ( $K_{\dot{\theta}} = 0.0$ ).

<b>Overall Performance</b>				
Total Length (Episodes)	6871			
Steady State Median Length (Steps)	627			
	Region 1	Region 2	Region 3	Region 4
Length (Episodes)	255	209	3204	3203
<b>Episode Length (Steps)</b>				
Mean	2383	769	660	644
Median	792	664	629	623
Standard Deviation	9884	260	124	101
Lower Quartile	634	605	589	585
Upper Quartile	1325	847	691	664
Minimum	520	530	517	513
Maximum	163839	3229	2447	3086

Table 6.28: KPIs for QL controller ( $K_{\dot{\theta}} = 0.0$ ).



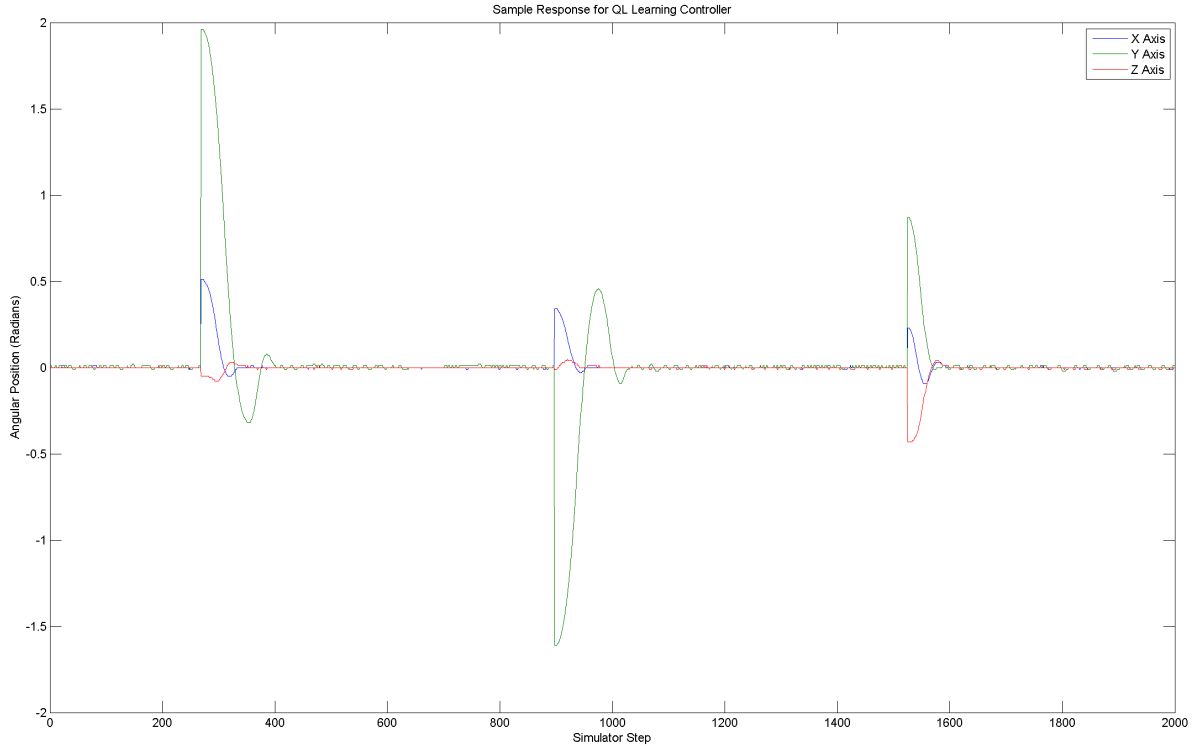


Figure 6.47: Sample time-domain response for QL controller ( $K_{\dot{\theta}} = 0.0$ ).

of the controller. In each experiment, this parameter was adjusted, whilst the remainder of the parameters remained fixed to values used previously, to allow comparison with previous experiments.

**High Resolution ( $\sigma_{\theta} = 256$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.29; the value of  $\sigma_{\theta}$  (the displacement ARBFN resolution) is increased to 256. The resulting smoothed set of episode lengths is shown in Figure 6.48.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.49. The key performance characteristics derived from the experiment are also shown in Table 6.30.

Comparison of these key performance values with those of the  $K_{\dot{\theta}} = 0.35$  experiment suggests that increasing the  $\sigma_{\theta}$  value results in no substantial change in steady-state performance. Steady-

Parameter	Value
$\epsilon$	0.05
$\alpha$	0.5
$\gamma$	0.9
$\lambda$	0.8
$\sigma_{\theta}$	256
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	0.35

Table 6.29: Tunable parameters for QL controller ( $\sigma_{\theta} = 256$ ).

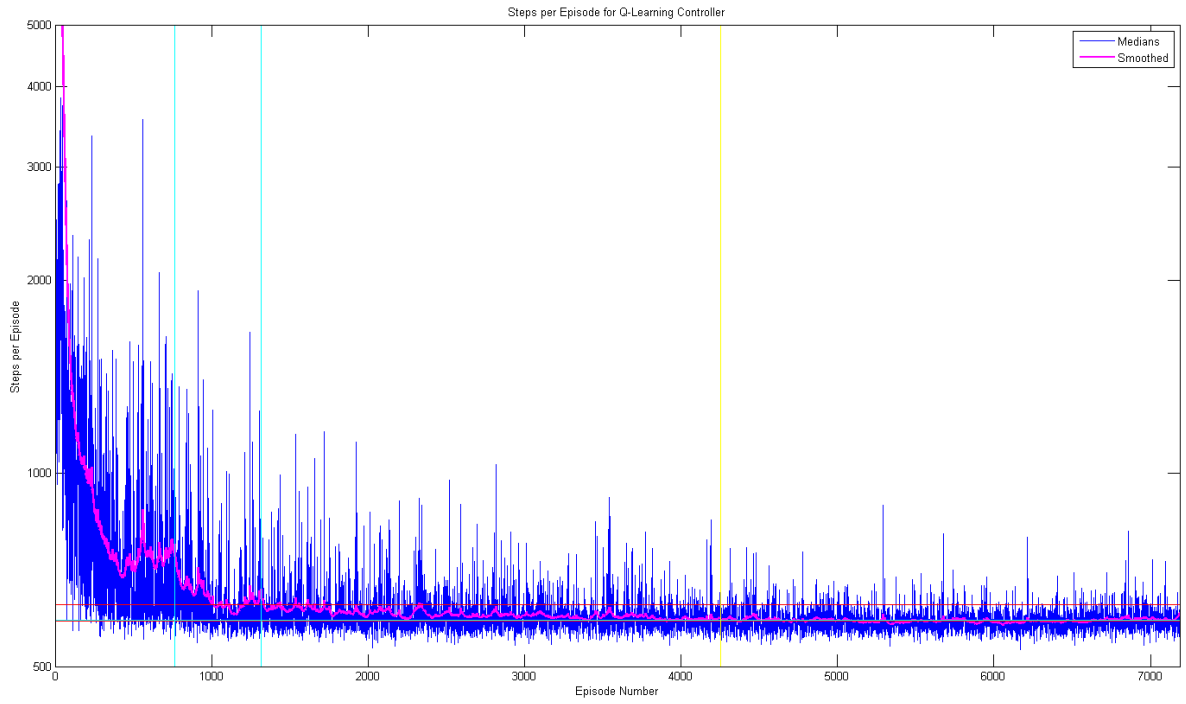


Figure 6.48: Smoothed episode lengths for QL controller ( $\sigma_\theta = 256$ ).

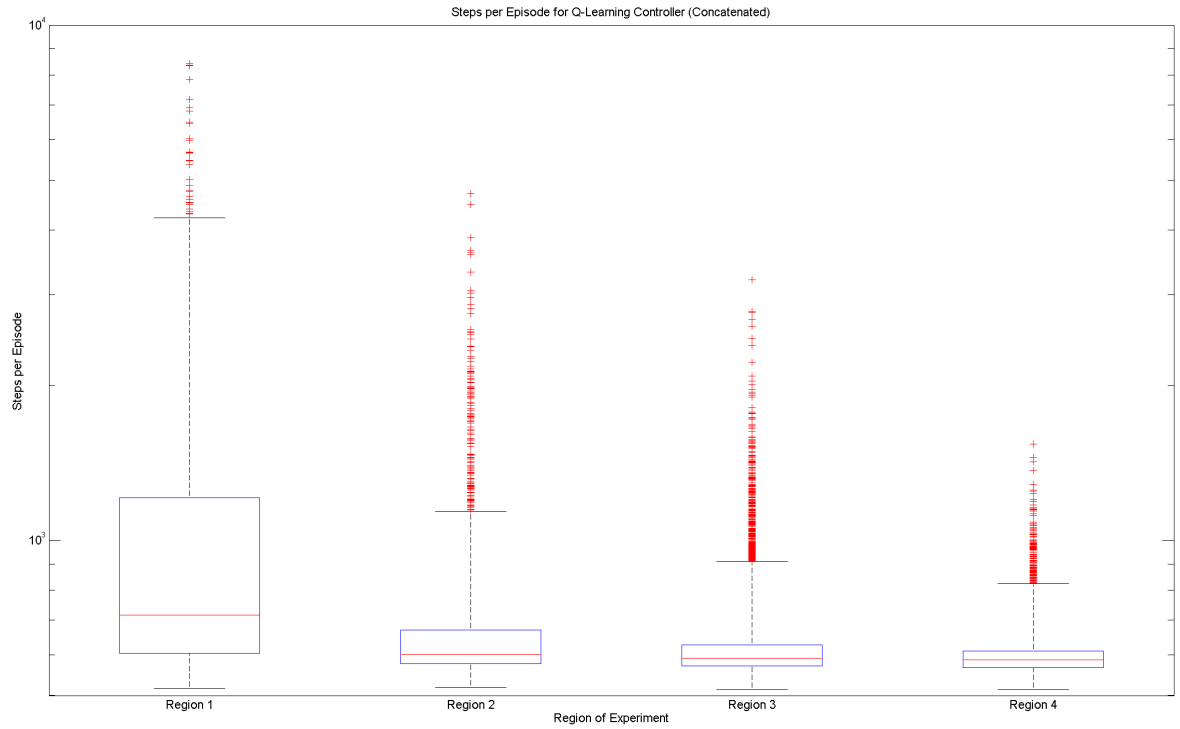


Figure 6.49: Distribution of concatenated episode lengths for QL controller ( $\sigma_\theta = 256$ ).

<b>Overall Performance</b>				
Total Length (Episodes)	7187			
Steady State Median Length (Steps)	590			
	Region 1	Region 2	Region 3	Region 4
Length (Episodes)	764	554	2935	2934
<b>Episode Length (Steps)</b>				
Mean	1196	732	632	601
Median	717	601	591	586
Standard Deviation	2834	394	147	67
Lower Quartile	604	576	570	567
Upper Quartile	1210	671	627	610
Minimum	516	518	513	513
Maximum	78674	4712	3200	1538

Table 6.30: KPIs for QL controller ( $\sigma_\theta = 256$ ).

state median episode length is identical to the  $\sigma_\theta = 128$  case, and standard deviation of episode lengths throughout the experiment are approximately similar. However, the learner does take longer to settle; there is a significant increase in the length of regions one and two, which also results in a reduction in total episodes completed.

Finally, consideration is also given to the time-domain response of the controller. A sample plot of the response is shown in Figure 6.50. The response of the controller appears similar to the  $\sigma_\theta = 128$  case; somewhat underdamped for large errors, but with oscillations arrested quickly.

**Low Resolution ( $\sigma_\theta = 80$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.31; the value of  $\sigma_\theta$  (the displacement ARBFN resolution) is reduced to 80. The resulting smoothed set of episode lengths is shown in Figure 6.51.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.52. The key performance characteristics derived from the experiment are also shown in Table 6.32.

Comparison of these key performance values with those of the  $K_{\dot{\theta}} = 0.35$  experiment suggests that reducing the  $\sigma_\theta$  value results in a noticeable reduction in performance throughout the experiment. Steady-state median episode length is increased slightly. More significantly however, the standard deviation of episode lengths is increased considerably. This instability means the learning behaviour fails to settle, which leads to a very large increase in the length of region two. However, it should be noted that the length of region one is reduced from the  $\sigma_\theta = 128$  case, indicating the learner initially learns faster.

Finally, consideration is also given to the time-domain response of the controller. A sample plot

Parameter	Value
$\epsilon$	0.05
$\alpha$	0.5
$\gamma$	0.9
$\lambda$	0.8
$\sigma_\theta$	80
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	0.35

Table 6.31: Tunable parameters for QL controller ( $\sigma_\theta = 80$ ).

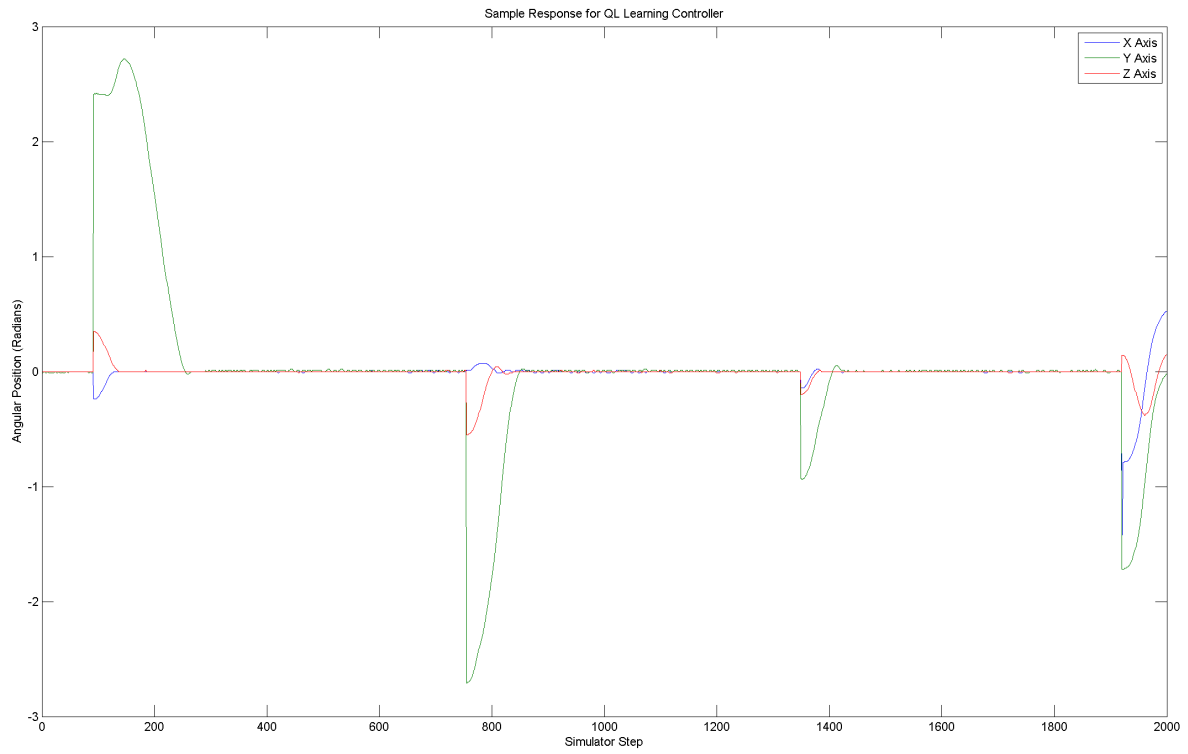


Figure 6.50: Sample time-domain response for QL controller ( $\sigma_\theta = 256$ ).

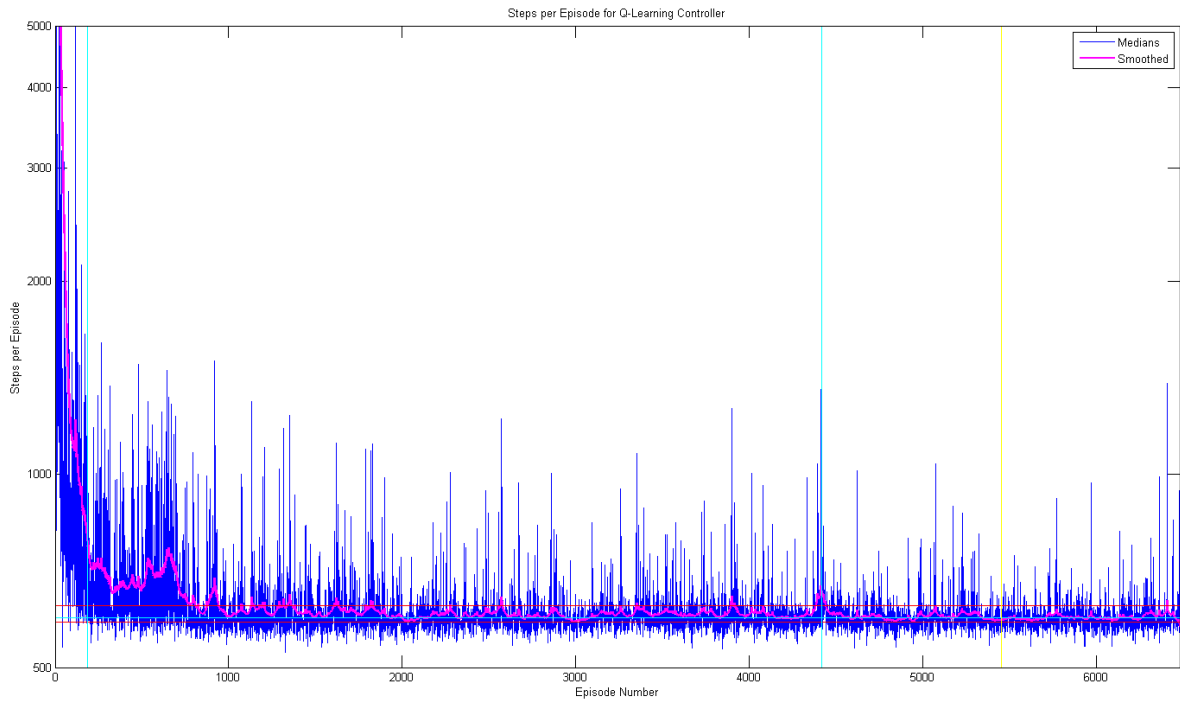


Figure 6.51: Smoothed episode lengths for QL controller ( $\sigma_\theta = 80$ ).

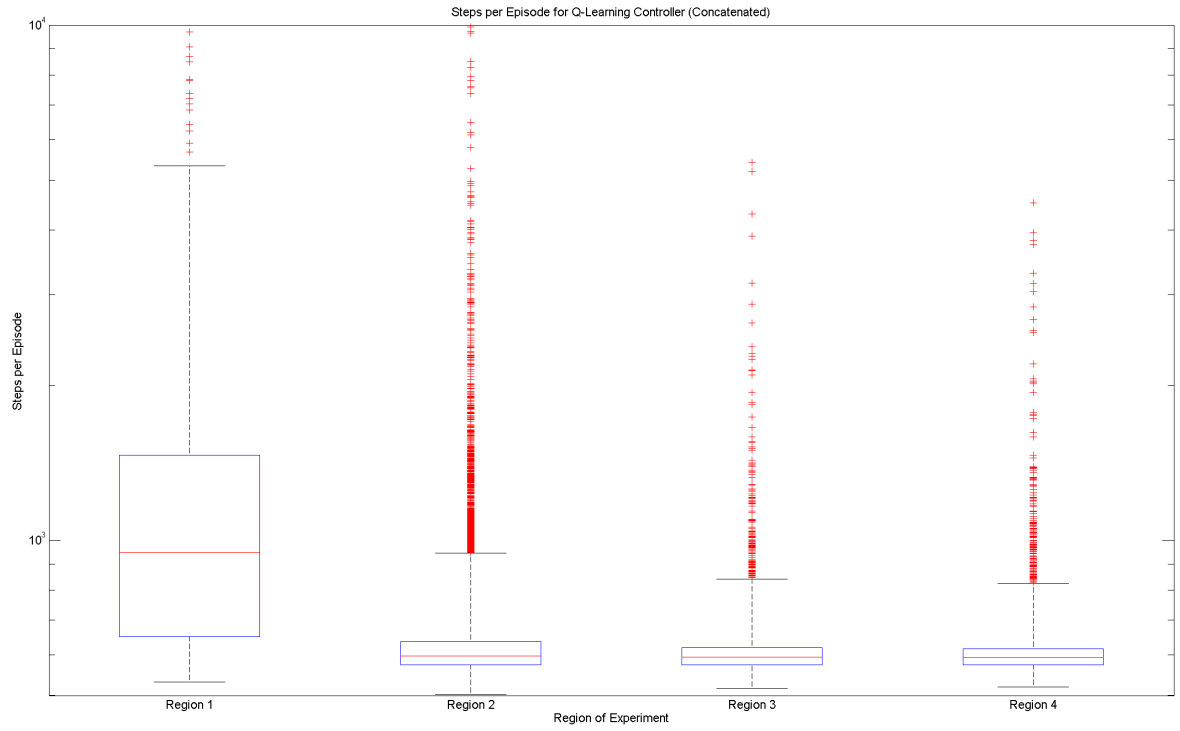


Figure 6.52: Distribution of concatenated episode lengths for QL controller ( $\sigma_\theta = 80$ ).

<b>Overall Performance</b>				
Total Length (Episodes)	6479			
Steady State Median Length (Steps)	597			
	Region 1	Region 2	Region 3	Region 4
Length (Episodes)	188	4230	1031	1030
<b>Episode Length (Steps)</b>				
Mean	1854	682	635	635
Median	947	596	593	593
Standard Deviation	3486	469	219	216
Lower Quartile	651	574	574	574
Upper Quartile	1462	636	619	616
Minimum	531	502	516	519
Maximum	31859	17515	5410	4516

Table 6.32: KPIs for QL controller ( $\sigma_\theta = 80$ ).

of the response is shown in Figure 6.53. The response of the controller remains underdamped, as for the  $\sigma_\theta = 128$  case, however the affects of the reduced resolution are clear. The reduced resolution results in jumpy behaviour whilst trying to settle at the nominal state. This can lead to increased episode length, if the quantisation error in the position leads to resetting of the settling time-out period.

**Low Resolution ( $\sigma_\theta = 32$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.33; the value of  $\sigma_\theta$  (the displacement ARBFN resolution) is reduced to 32.

The results from these trials are substantially different to those seen previously; the controller fails to settle to a reasonably stable performance level; episode length fluctuates wildly, from about 600 steps in length, up to more than 10,000, throughout the trials. With performance this poor, further analysis is irrelevant. The set of smoothed episode lengths is shown in Figure 6.54 to illustrate the poor performance.

The most likely cause of this degenerate performance is that the resolution of the ARBFN is now so low, as to result in the situation where the controller is unable to distinguish states which are sufficiently close to the nominal state to satisfy the settling criteria, from those which do not satisfy the settling criteria. This means the controller will often not stabilise the vehicle sufficiently close to the desired position for the settling time-out to be activated, which results in the episode not ending even though the controller has brought the vehicle to a halt.

This degenerate situation must obviously be avoided in any real control system.

The experiments show that whilst there are significant performance consequences from having a resolution which is too low, increasing the resolution does not continue to offer improvements in performance beyond some threshold. Additionally, increasing the resolution results in increased learning time, and additionally results in increased computational load (see §7 for further information). Hence, it is probably preferable simply to select the minimum resolution which is known to offer adequate performance.

The value of  $\sigma_\theta = 128$  as used originally appears to offer adequate performance, so further experiments will continue to use this same value.

### 6.2.7 Varying Velocity ARBFN Resolution $\sigma_{\dot{\theta}}$

A set of experiments were then conducted to characterise the effect which variations in angular velocity resolution parameter  $\sigma_{\dot{\theta}}$  for the Q-Function estimator would have on the performance of the controller. In each experiment, this parameter was adjusted, whilst the remainder of

Parameter	Value
$\epsilon$	0.05
$\alpha$	0.5
$\gamma$	0.9
$\lambda$	0.8
$\sigma_\theta$	32
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	0.35

Table 6.33: Tunable parameters for QL controller ( $\sigma_\theta = 32$ ).

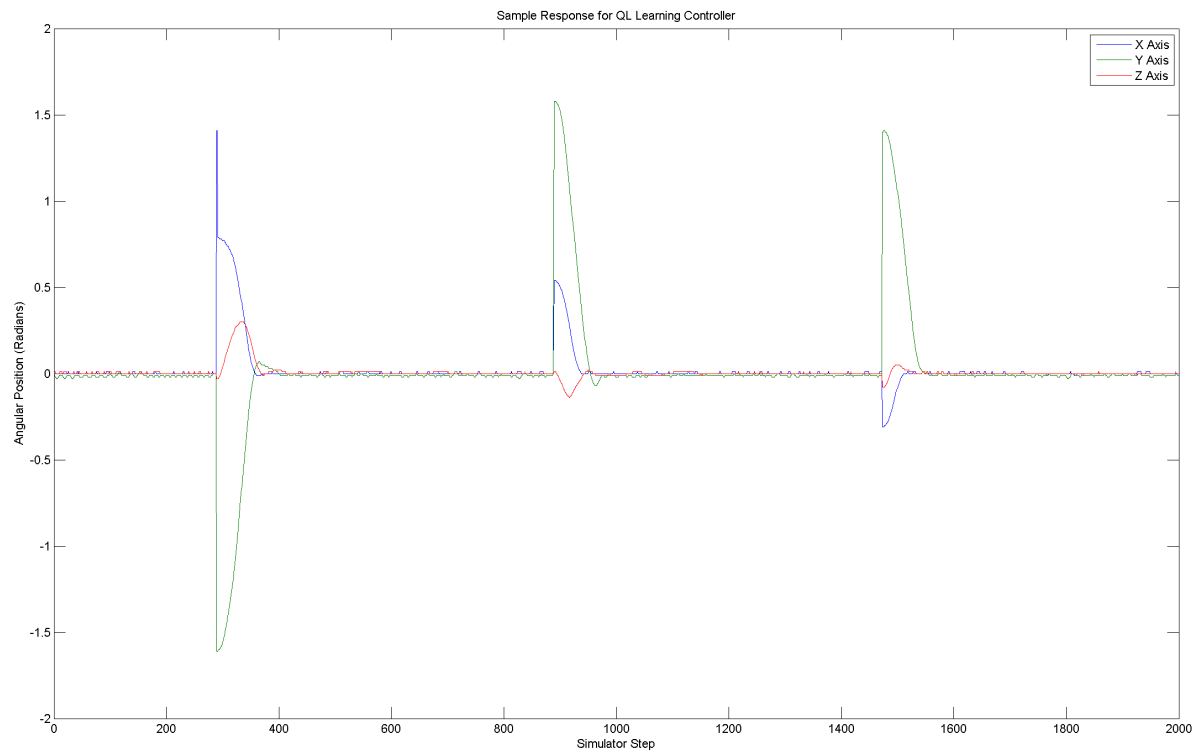


Figure 6.53: Sample time-domain response for QL controller ( $\sigma_\theta = 80$ ).

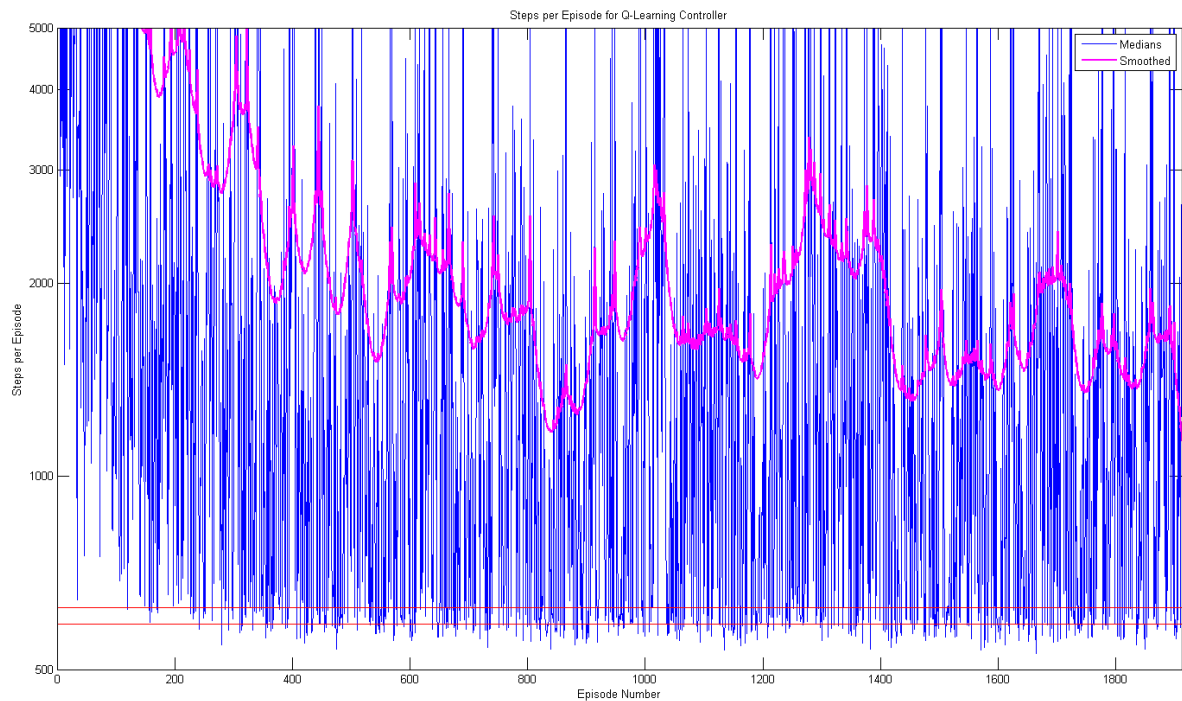


Figure 6.54: Smoothed episode lengths for QL controller ( $\sigma_\theta = 32$ ).

the parameters remained fixed to values used previously, to allow comparison with previous experiments.

**High Resolution ( $\sigma_{\dot{\theta}} = 128$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.34; the value of  $\sigma_{\dot{\theta}}$  (the velocity ARBFN resolution) is increased to 128. The resulting smoothed set of episode lengths is shown in Figure 6.55.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.56. The key performance characteristics derived from the experiment are also shown in Table 6.35.

Comparison of these key performance values with those of the  $K_{\dot{\theta}} = 0.35$  experiment suggests that increasing the  $\sigma_{\dot{\theta}}$  values across all axes results in a slight reduction in overall performance. There is a slight reduction in steady-state median episode length, though not significantly. There is an increase in standard deviation, particularly in regions three and four. Most significantly, there is a reduction in the performance of the learner, which takes considerably longer to settle, as evidenced by the increase in lengths of regions one and two.

Finally, consideration is also given to the time-domain response of the controller. A sample plot of the response is shown in Figure 6.57. The response of the controller appears similar to the  $\sigma_{\dot{\theta}} = 64$  case; somewhat underdamped for large errors, but with oscillations arrested quickly.

**Low Resolution ( $\sigma_{\dot{\theta}} = 32$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.36; the value of  $\sigma_{\dot{\theta}}$  (the velocity ARBFN resolution) is reduced to 32. The resulting smoothed set of episode lengths is shown in Figure 6.58.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.59. The key performance characteristics derived from the experiment are also shown in Table 6.37.

Comparison of these key performance values with those of the  $K_{\dot{\theta}} = 0.35$  experiment suggests that reducing the  $\sigma_{\dot{\theta}}$  value results in general improvements in performance throughout the experiment. Median episode length remains the same throughout the experiment, but the learner settles more quickly, as evidenced by a reduction in the lengths of regions one and two. However, there is a considerable increase in the standard deviation of episode lengths during regions one, two and three (probably this is due to the presence of some very high outlying values).

Finally, consideration is also given to the time-domain response of the controller. A sample plot of the response is shown in Figure 6.60. The response of the controller appears similar

Parameter	Value
$\epsilon$	0.05
$\alpha$	0.5
$\gamma$	0.9
$\lambda$	0.8
$\sigma_{\theta}$	128
$\sigma_{\dot{\theta}}$	128
$K_{\dot{\theta}}$	0.35

Table 6.34: Tunable parameters for QL controller ( $\sigma_{\dot{\theta}} = 128$ ).



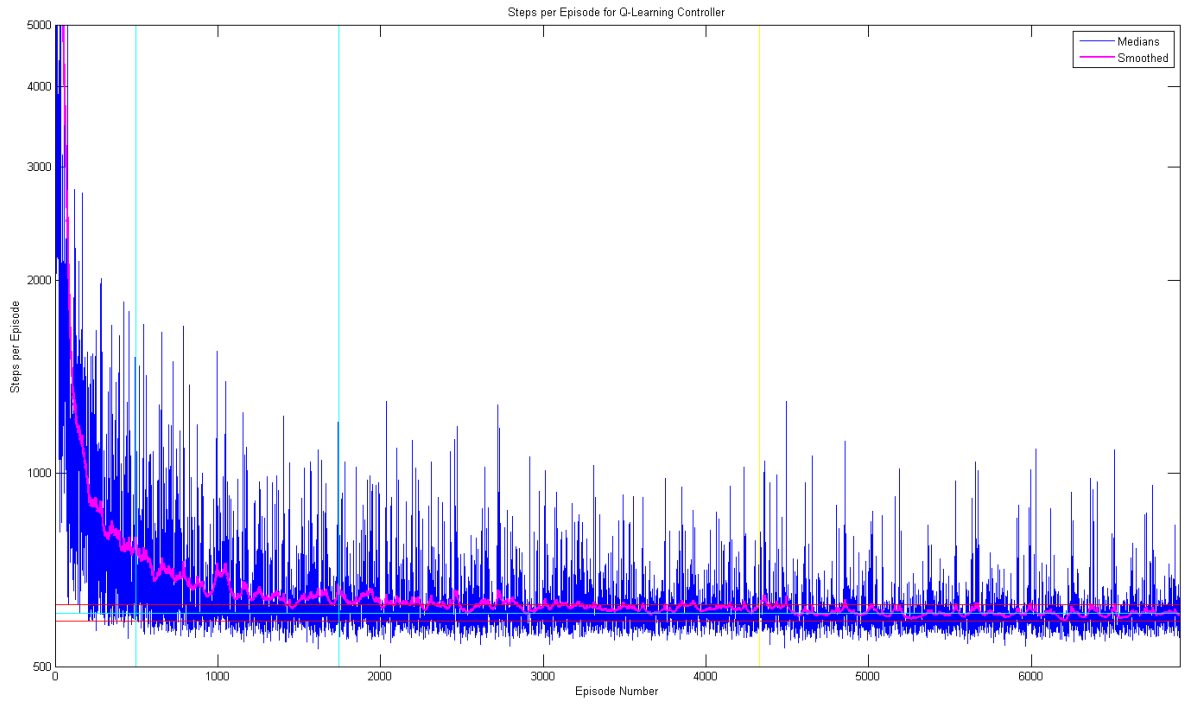


Figure 6.55: Smoothed episode lengths for QL controller ( $\sigma_{\theta} = 128$ ).

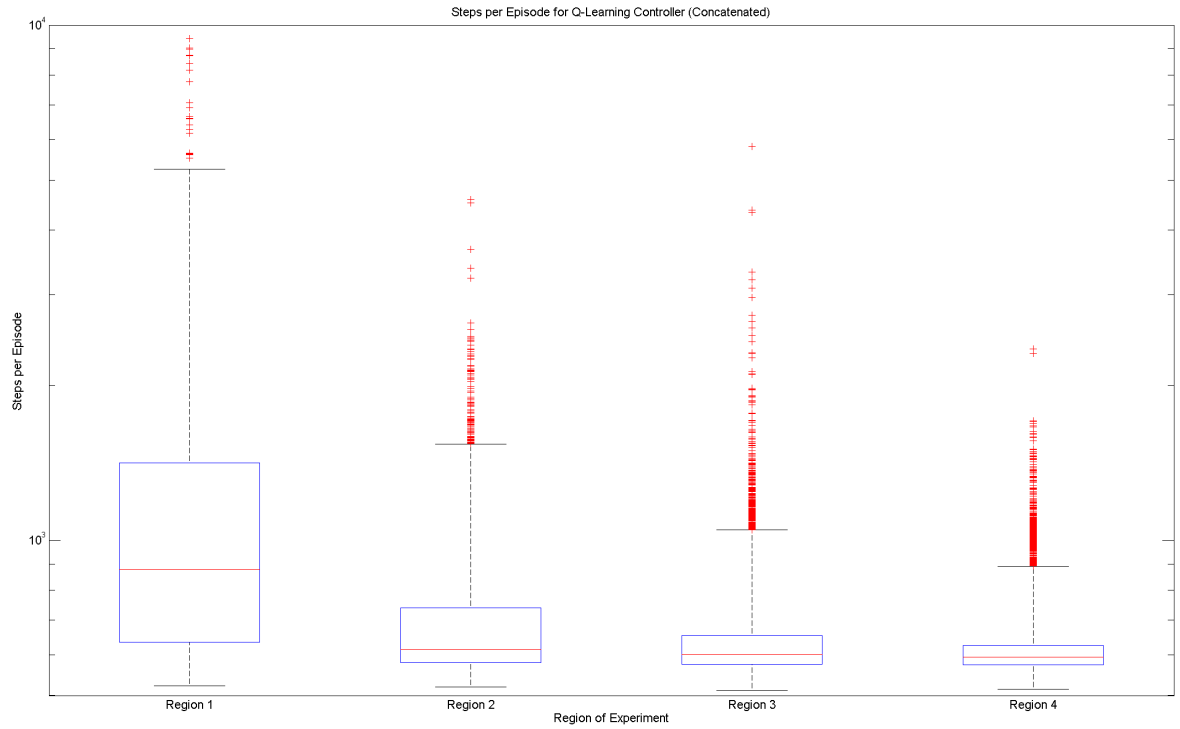


Figure 6.56: Distribution of concatenated episode lengths for QL controller ( $\sigma_{\theta} = 128$ ).

<b>Overall Performance</b>				
Total Length (Episodes)	6913			
Steady State Median Length (Steps)	605			
	<b>Region 1</b>	<b>Region 2</b>	<b>Region 3</b>	<b>Region 4</b>
Length (Episodes)	498	1245	2585	2585
<b>Episode Length (Steps)</b>				
Mean	1456	728	660	632
Median	878	615	601	594
Standard Deviation	3752	297	199	128
Lower Quartile	635	579	575	573
Upper Quartile	1414	739	654	626
Minimum	522	519	512	514
Maximum	84816	4583	5809	2354

Table 6.35: KPIs for QL controller ( $\sigma_{\dot{\theta}} = 128$ ).

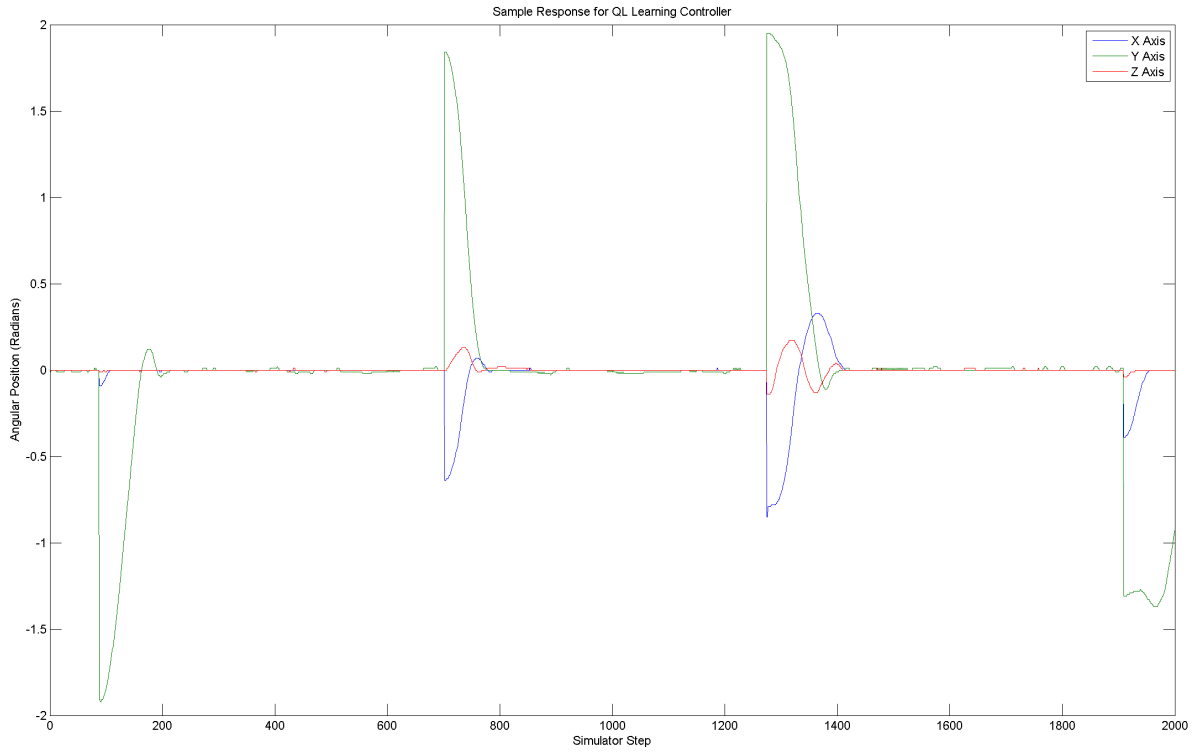


Figure 6.57: Sample time-domain response for QL controller ( $\sigma_{\dot{\theta}} = 128$ ).

Parameter	Value
$\epsilon$	0.05
$\alpha$	0.5
$\gamma$	0.9
$\lambda$	0.8
$\sigma_{\theta}$	80
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	0.35

Table 6.36: Tunable parameters for QL controller ( $\sigma_{\dot{\theta}} = 32$ ).

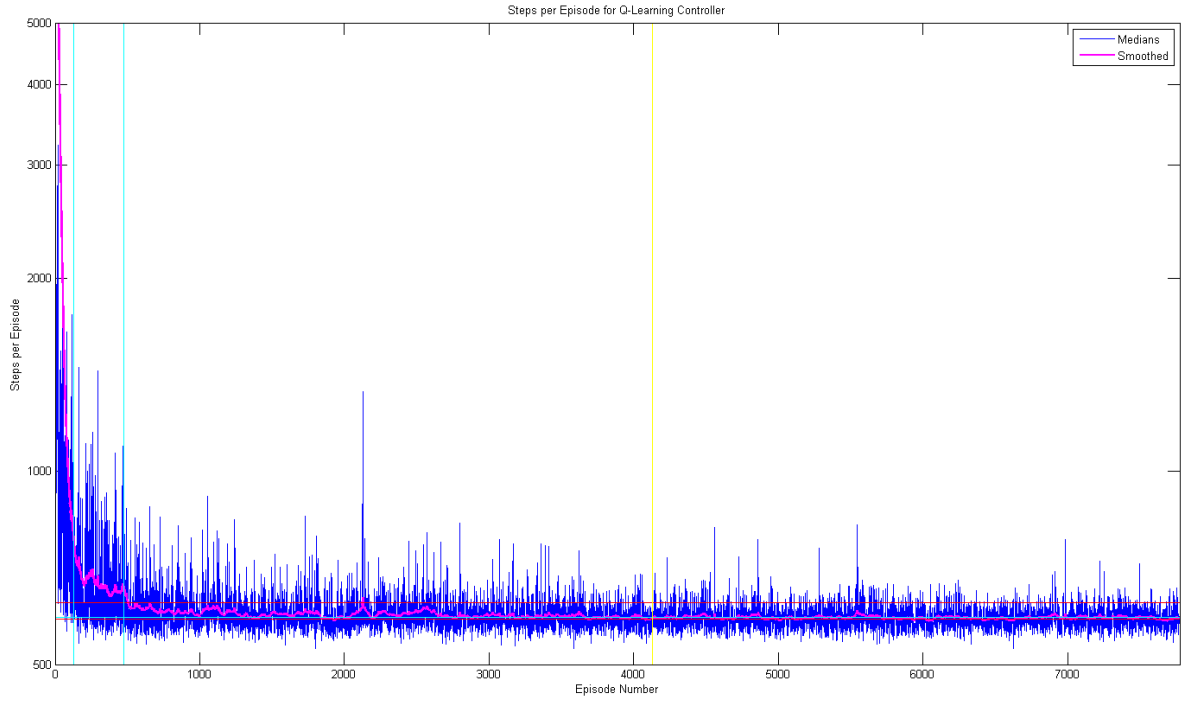


Figure 6.58: Smoothed episode lengths for QL controller ( $\sigma_{\dot{\theta}} = 32$ ).

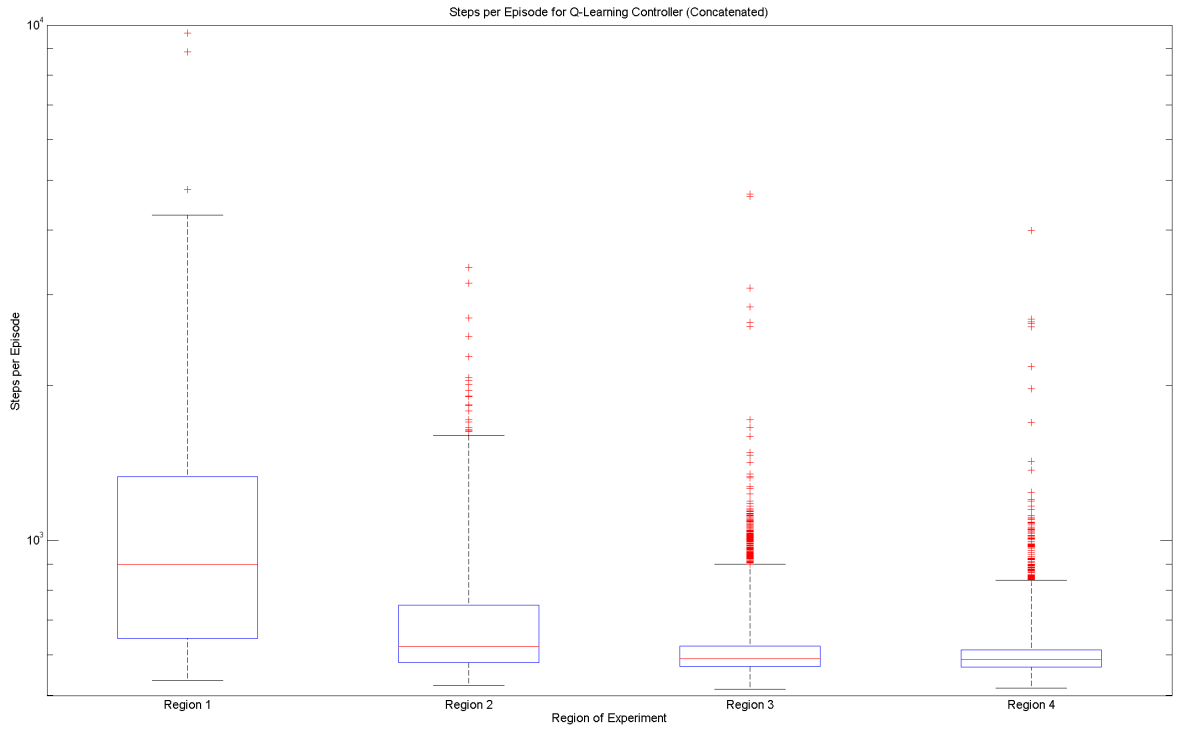


Figure 6.59: Distribution of concatenated episode lengths for QL controller ( $\sigma_{\dot{\theta}} = 32$ ).

<b>Overall Performance</b>				
Total Length (Episodes)	7774			
Steady State Median Length (Steps)	592			
	Region 1	Region 2	Region 3	Region 4
Length (Episodes)	130	347	3649	3648
<b>Episode Length (Steps)</b>				
Mean	1536	878	620	600
Median	897	622	590	588
Standard Deviation	3732	5008	747	79
Lower Quartile	646	579	569	588
Upper Quartile	1330	749	624	613
Minimum	535	523	514	517
Maximum	58900	162289	76886	3988

Table 6.37: KPIs for QL controller ( $\sigma_{\dot{\theta}} = 32$ ).

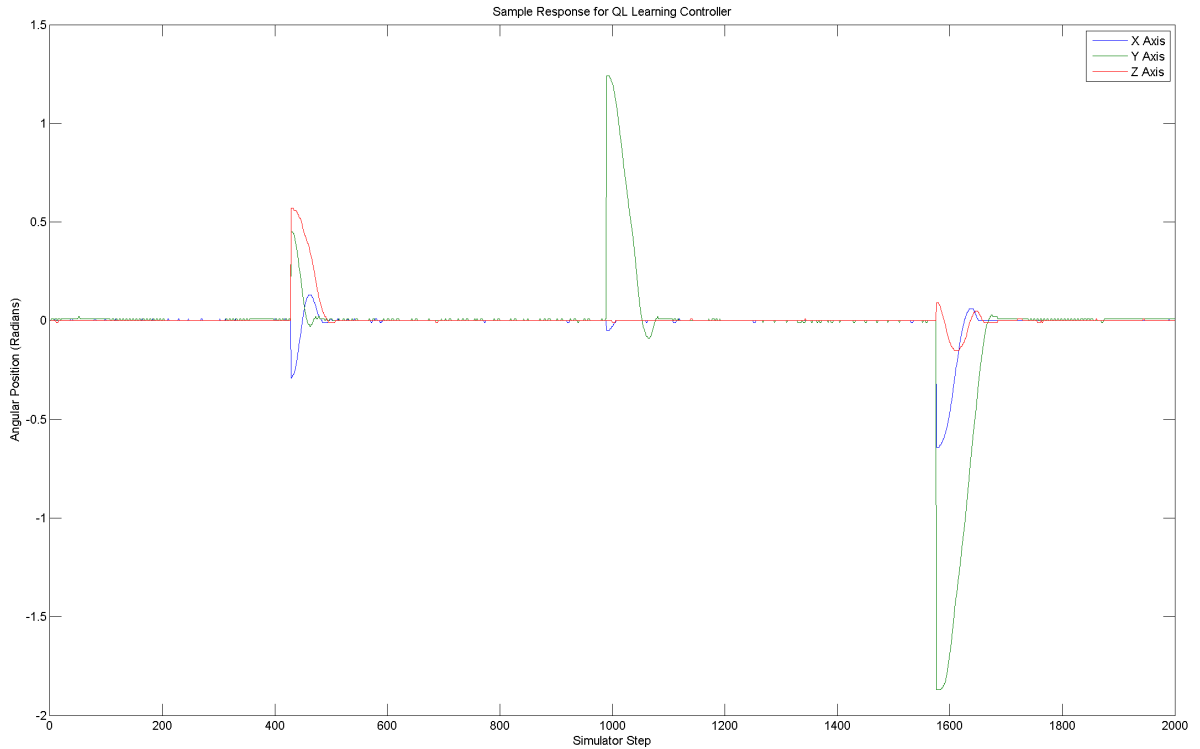


Figure 6.60: Sample time-domain response for QL controller ( $\sigma_{\dot{\theta}} = 32$ ).

to the  $\sigma_{\dot{\theta}} = 64$  case; somewhat underdamped for large errors, but with oscillations arrested quickly.

The experiments indicate that there are some improvements in performance possible by selecting appropriate resolution parameters. However, largely the results agree with those from §6.2.6: that whilst selecting a value which is too low may result in catastrophically bad performance, selecting higher values will simply return an increase in required learning time, without any improvement in control performance. Hence, it is probably preferable simply to select the minimum resolution which is known to offer adequate performance. Whilst the value of  $\sigma_{\dot{\theta}} = 32$  appeared to offer some performance improvements over the original value of  $\sigma_{\dot{\theta}} = 64$ , the substantial increase in standard deviation of episode lengths suggests that there could be stability issues. Hence, further experiments will continue to use the original value of  $\sigma_{\dot{\theta}} = 64$ .

### 6.2.8 Varying Greediness $\epsilon$

A set of experiments were then conducted to characterise the effect which variations in the controller parameter  $\epsilon$  would have on the performance of the controller. In each experiment, this parameter was adjusted, whilst the remainder of the parameters remained fixed to values used previously, to allow comparison with previous experiments.

**High Epsilon ( $\epsilon = 0.2$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.38; the value of  $\epsilon$  (the greediness parameter) is increased to 0.2. The resulting smoothed set of episode lengths is shown in Figure 6.61.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.62. The key performance characteristics derived from the experiment are also shown in Table 6.39.

Comparison of these key performance values with those of the  $K_{\dot{\theta}} = 0.35$  experiment suggests that the increase in  $\epsilon$  value results in significantly degraded performance in most respects. The learner fails to settle adequately, leading in region two being substantially longer. The median episode length is greater in all regions except for region one, whilst the standard deviation is considerably higher in regions three and four. The less greedy behaviour of the controller (which selects exploratory actions with greater probability) means that learning initially progresses faster, as the controller is more likely to stumble upon a favourable behaviour. However, the presence of exploratory actions prevents the controller from being able to operate stably, with sub-optimally long episodes continuing to occur throughout the experiment.

Finally, consideration is also given to the time-domain response of the controller. A sample plot of the response is shown in Figure 6.63. The response of the controller remains similar to

Parameter	Value
$\epsilon$	0.20
$\alpha$	0.2
$\gamma$	0.9
$\lambda$	0.8
$\sigma_{\theta}$	128
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	0.35

Table 6.38: Tunable parameters for QL controller ( $\epsilon = 0.2$ ).

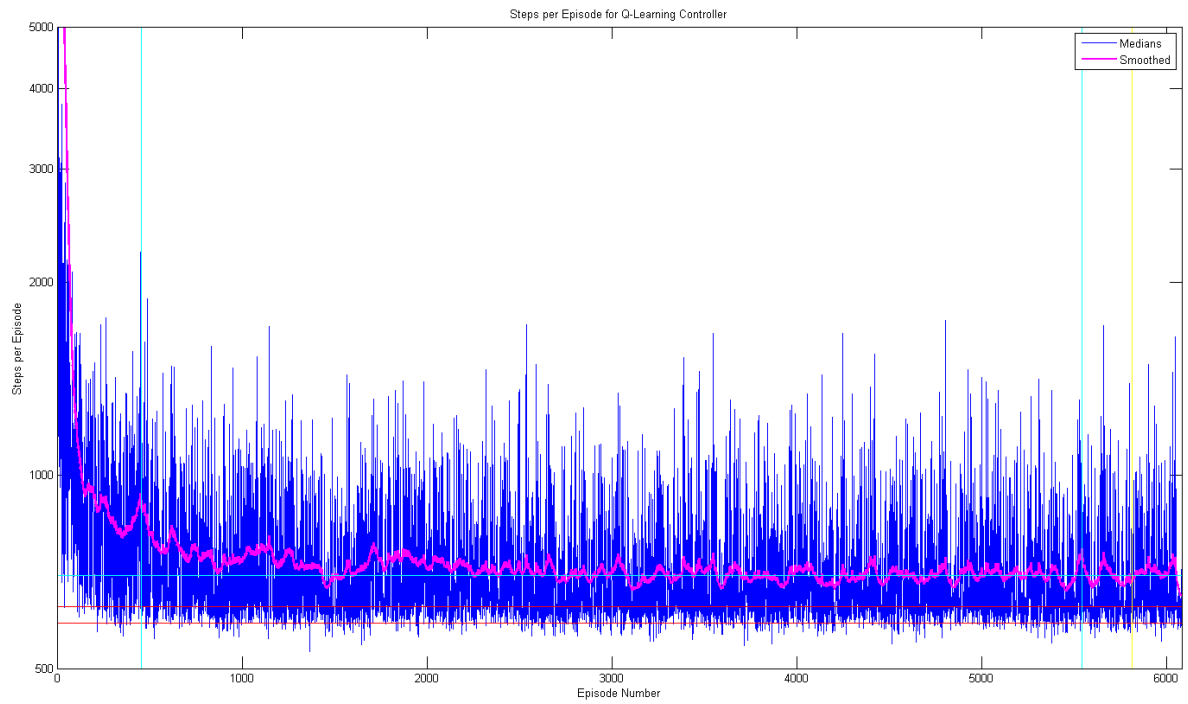


Figure 6.61: Smoothed episode lengths for QL controller ( $\epsilon = 0.2$ ).

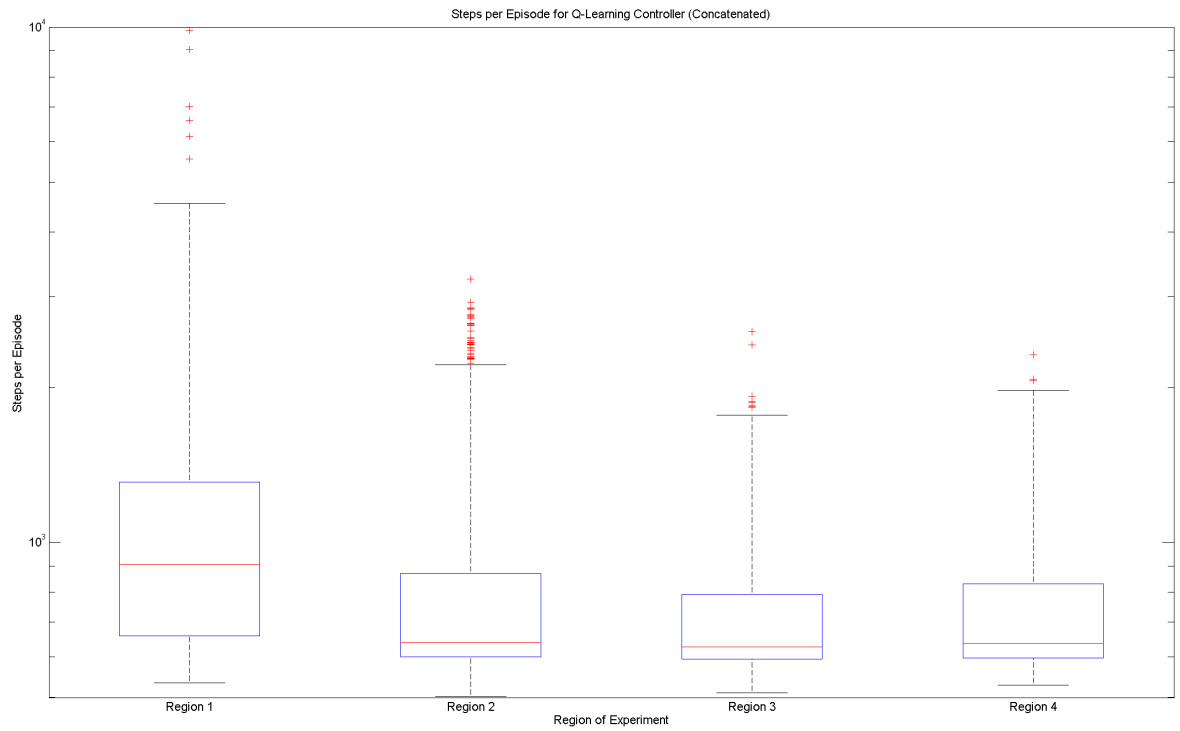


Figure 6.62: Distribution of concatenated episode lengths for QL controller ( $\epsilon = 0.2$ ).

<b>Overall Performance</b>				
Total Length (Episodes)	6080			
Steady State Median Length (Steps)	698			
	Region 1	Region 2	Region 3	Region 4
Length (Episodes)	457	5082	271	270
<b>Episode Length (Steps)</b>				
Mean	1235	770	747	753
Median	906	640	628	636
Standard Deviation	2348	275	265	258
Lower Quartile	658	599	594	597
Upper Quartile	1309	870	799	831
Minimum	534	502	511	528
Maximum	50815	3244	2564	2312

Table 6.39: KPIs for QL controller ( $\epsilon = 0.2$ ).

that of the  $\epsilon = 0.05$  case. The response of all three axes are underdamped; there is substantial initial overshoot in the step response for large errors, though the response decays immediately following this. However, there is some instability visible after the response has settled at the nominal position, caused by the increased frequency with which exploratory actions (which may disturb the controller away from the nominal state) occur.

**Low Epsilon ( $\alpha = 0.02$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.42; the value of  $\epsilon$  (the greediness parameter) is decreased to 0.02. This set of parameters delivers the best performance obtained during these parameter tuning experiments; accordingly, this experiment becomes the final baseline against which performance of the controller is measured. Complete details are listed in §6.2.9.

**Low Epsilon ( $\alpha = 0.01$ )** An experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.40; the value of  $\epsilon$  (the greediness parameter) is decreased to 0.01. The resulting smoothed set of episode lengths is shown in Figure 6.64.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.65. The key performance characteristics derived from the experiment are also shown in Table 6.41.

Comparison of these key performance values with those of the  $K_{\dot{\theta}} = 0.35$  experiment suggests that the reduction in  $\epsilon$  value results in a noticeable improvement in steady-state performance. The median episode length is reduced slightly throughout the experiment. Standard deviation is higher in regions one through three (although the standard deviation in region three is probably substantially affected by outliers). Due to the reduced affect of exploratory actions, the learner takes longer to find suitable behaviour, leading to an increase in the length of region one. However, once suitable behaviour to control the environment is discovered, the controller stabilizes

Parameter	Value
$\epsilon$	0.01
$\alpha$	0.2
$\gamma$	0.9
$\lambda$	0.8
$\sigma_{\theta}$	128
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	0.35

Table 6.40: Tunable parameters for QL controller ( $\epsilon = 0.01$ ).

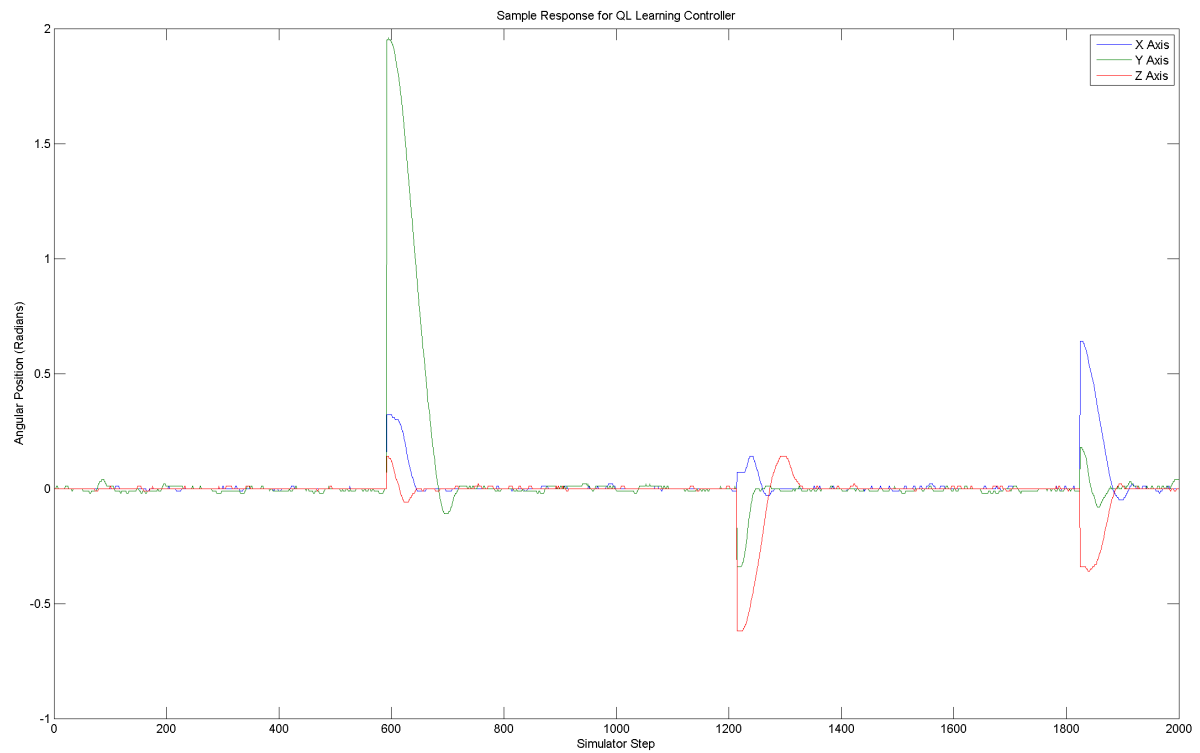


Figure 6.63: Sample time-domain response for QL controller ( $\epsilon = 0.2$ ).

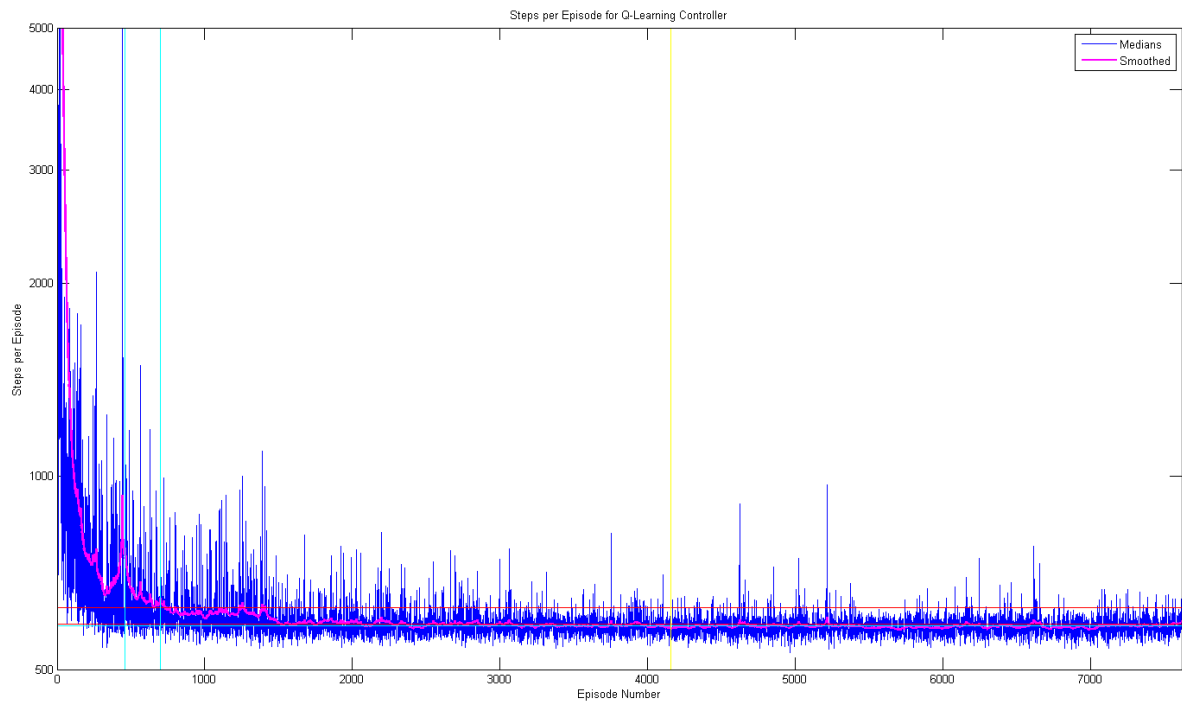


Figure 6.64: Smoothed episode lengths for QL controller ( $\epsilon = 0.01$ ).



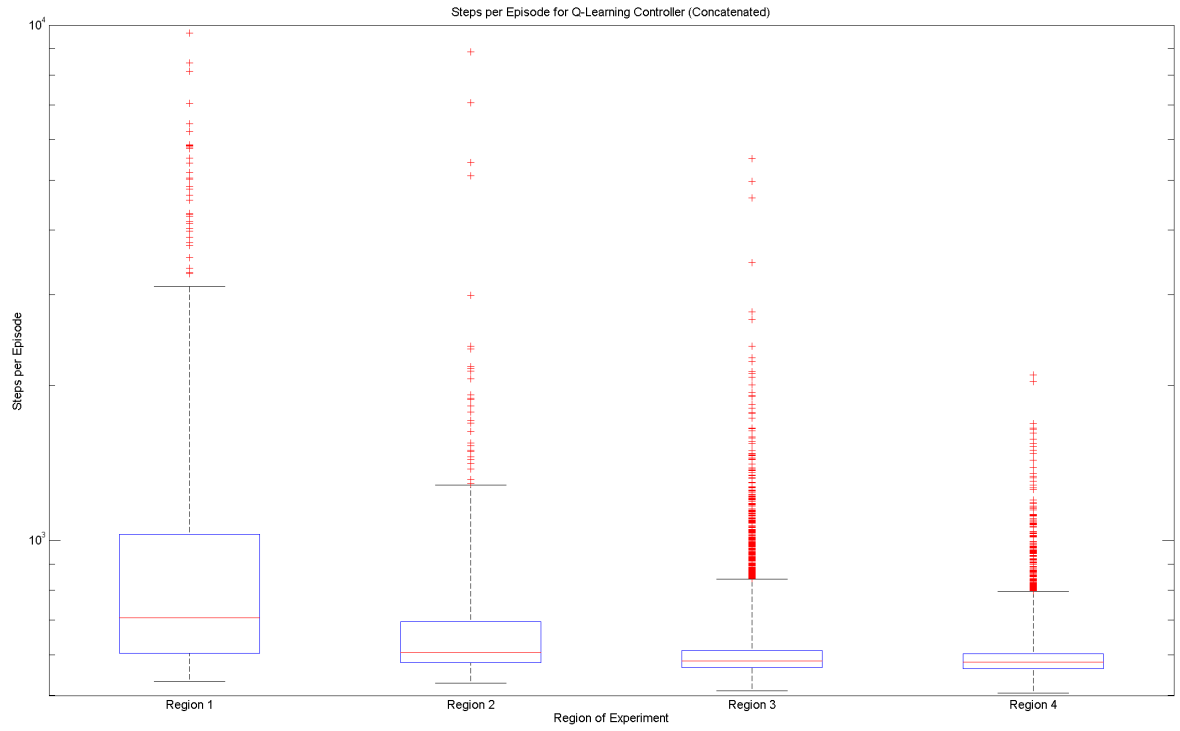


Figure 6.65: Distribution of concatenated episode lengths for QL controller ( $\epsilon = 0.01$ ).

<b>Overall Performance</b>				
Total Length (Episodes)	7617			
Steady State Median Length (Steps)	584			
	Region 1	Region 2	Region 3	Region 4
Length (Episodes)	461	240	3458	3458
<b>Episode Length (Steps)</b>				
Mean	1238	740	620	593
Median	707	606	584	581
Standard Deviation	3151	528	394	69
Lower Quartile	604	579	566	563
Upper Quartile	1029	699	612	602
Minimum	533	528	510	506
Maximum	72982	8867	28449	2094

Table 6.41: KPIs for QL controller ( $\epsilon = 0.01$ ).

quickly, with a reduction in the length of region two. Overall, the increase in the length of region one is not fully compensated for by the reduction in median episode length, leading to a slight reduction in total episodes completed.

Finally, consideration is also given to the time-domain response of the controller. A sample plot of the response is shown in Figure 6.66. The response of the controller remains similar to that of the  $\epsilon = 0.05$  case. The response of all three axes are underdamped; there is substantial initial overshoot in the step response for large errors, though the response decays immediately following this. The post-settling instability present in the  $\epsilon = 0.2$  case is not present (as expected).

The experiments show that there is a clear trade-off in selecting a value for  $\epsilon$ , between faster initial learning (and presumably ability to deal with local minima), and more stable steady-state behaviour.

### 6.2.9 Final Baseline

A final experiment consisting of three trials was performed, using the set of controller parameters shown in Table 6.42. Since these parameter values were optimised in the preceding series of experiments, the performance of this final experiment should be representative of the level of performance which can be attained from this design of controller, given suitable parameter selection.

The resulting episode lengths for each trial are plotted in Figure 6.67. Then, an identical method for collating and analysing the results from each trial was followed as used in §6.2.1. The resulting smoothed set of episode lengths is shown in Figure 6.68.

Figures 6.69, 6.70, 6.71, and 6.72 show the distribution of episode lengths in each region of the experiment, for the individual trials, and also the median and concatenated data sets in that region.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.73. The key performance characteristics derived from the experiment are also shown in Table 6.43.

Comparison of these key performance values with those of the  $K_{\dot{\theta}} = 0.35$  experiment detailed in §6.2.5 suggests that the reduction in  $\epsilon$  value (from  $\epsilon = 0.2$  in that experiment) results in minimal change in performance. There is a very slight reduction in median episode length in regions three and four. The first region is less stable, with a significantly higher standard deviation of episode lengths, and is accordingly longer. However, behaviour in the second region is more stable, with significantly lower standard deviation, and shorter length.

Parameter	Value
$\epsilon$	0.02
$\alpha$	0.2
$\gamma$	0.9
$\lambda$	0.8
$\sigma_{\theta}$	128
$\sigma_{\dot{\theta}}$	64
$K_{\dot{\theta}}$	0.35

Table 6.42: Final set of tunable parameters for QL controller.

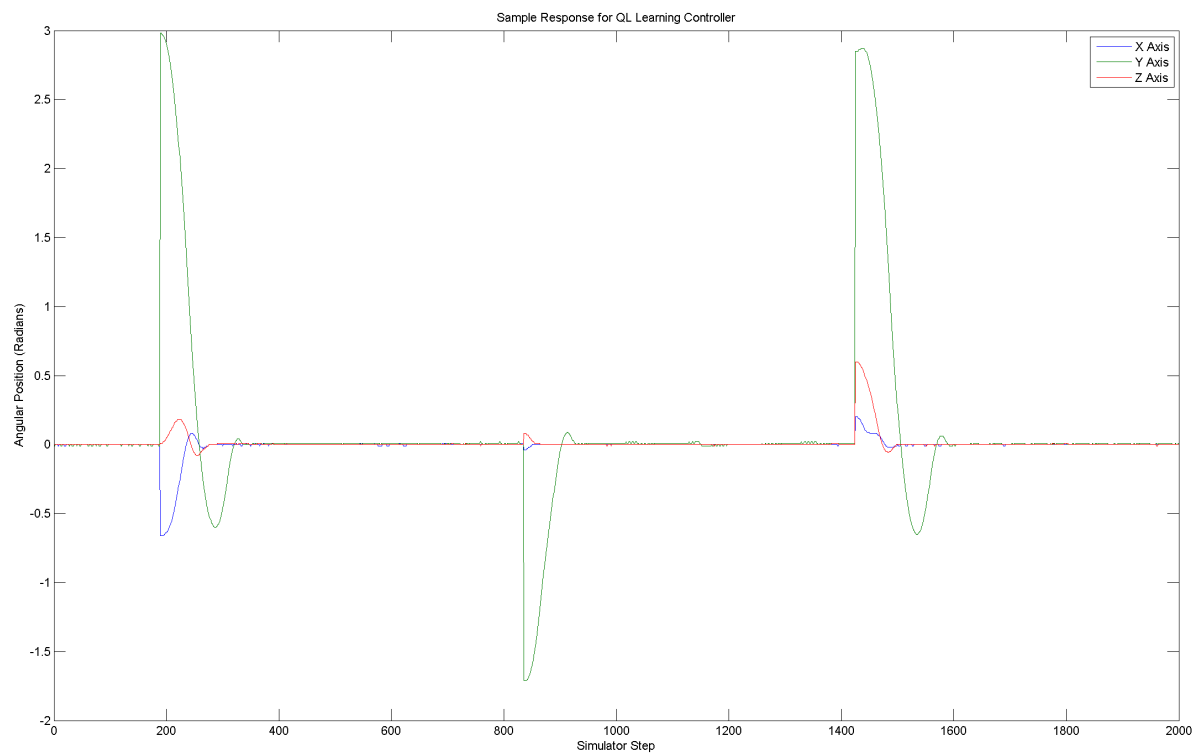


Figure 6.66: Sample time-domain response for QL controller ( $\epsilon = 0.01$ ).

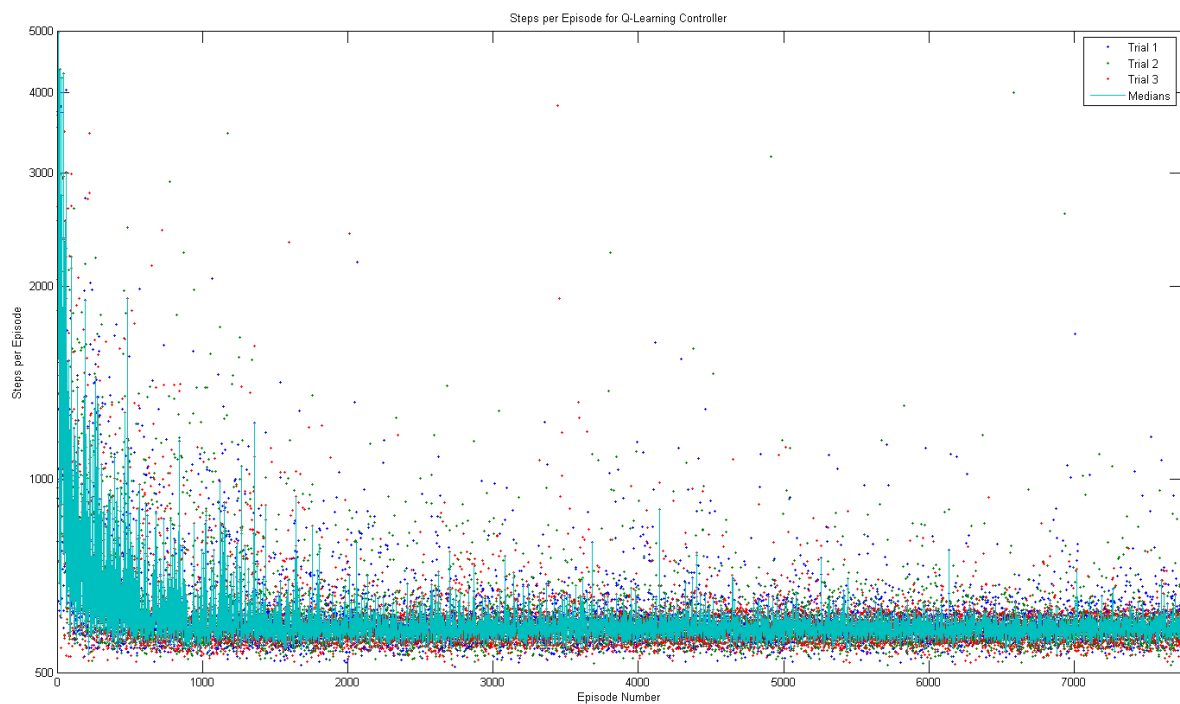


Figure 6.67: Individual episode lengths for QL controller (final baseline).

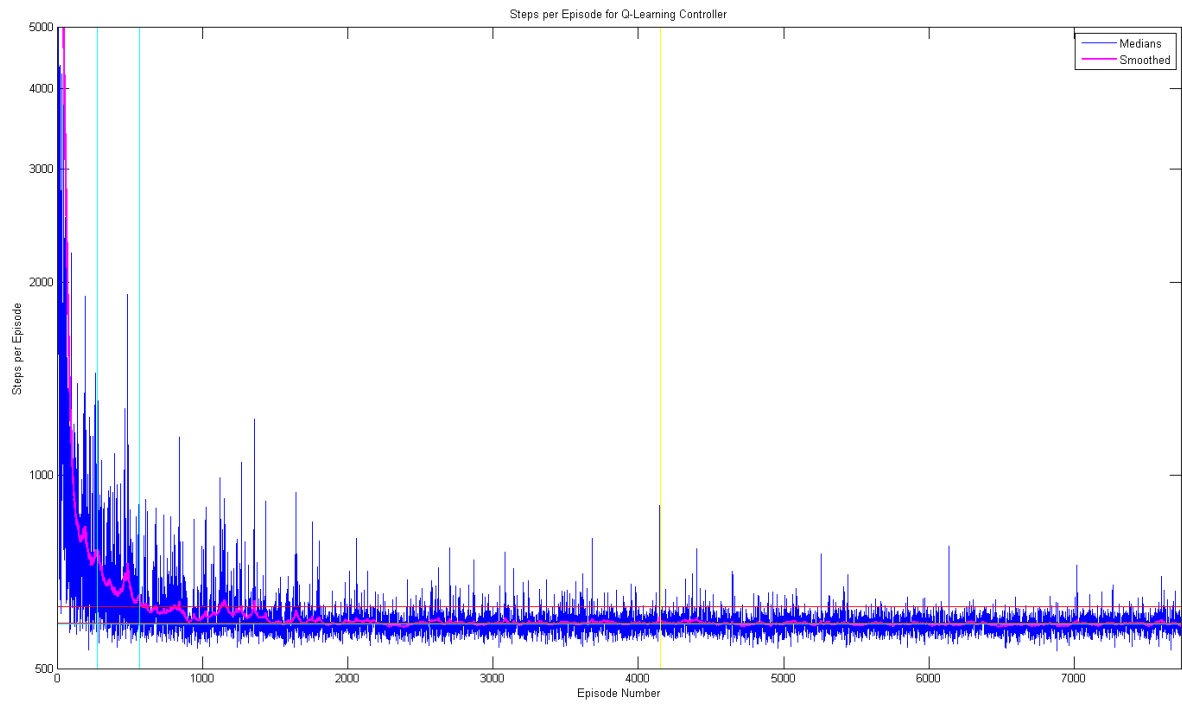


Figure 6.68: Smoothed episode lengths for QL controller (final baseline).

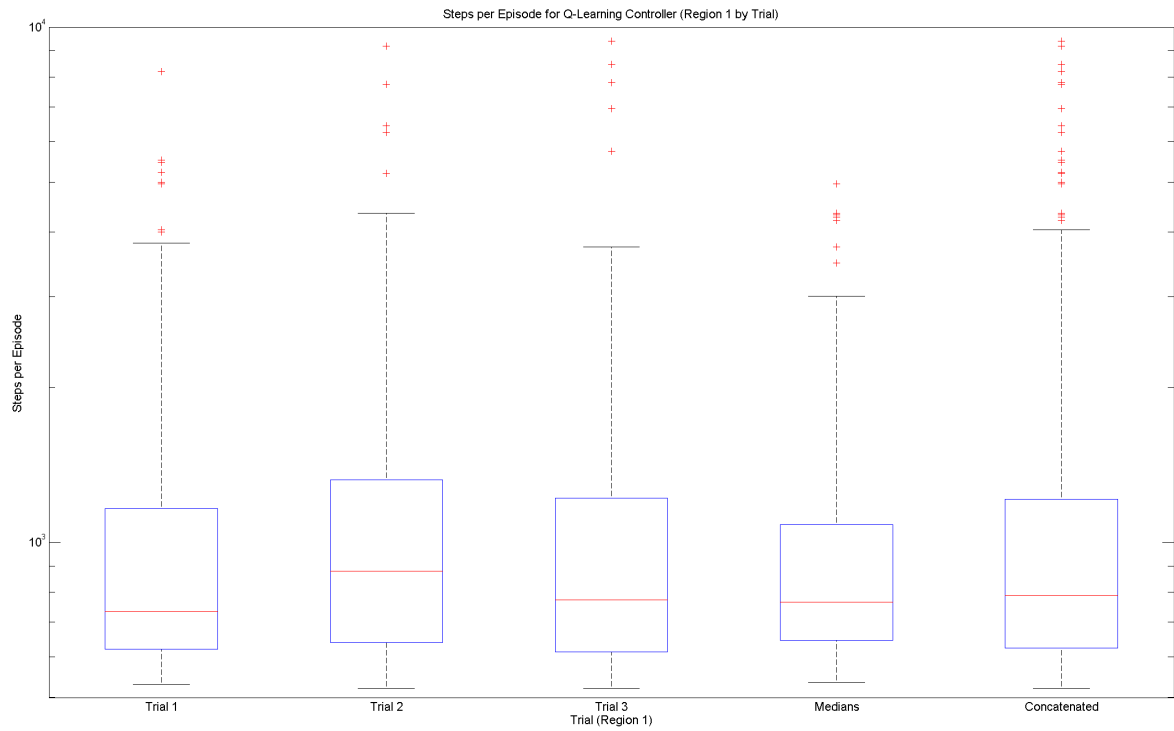


Figure 6.69: Distribution of episode lengths in Region 1 for QL controller (final baseline).

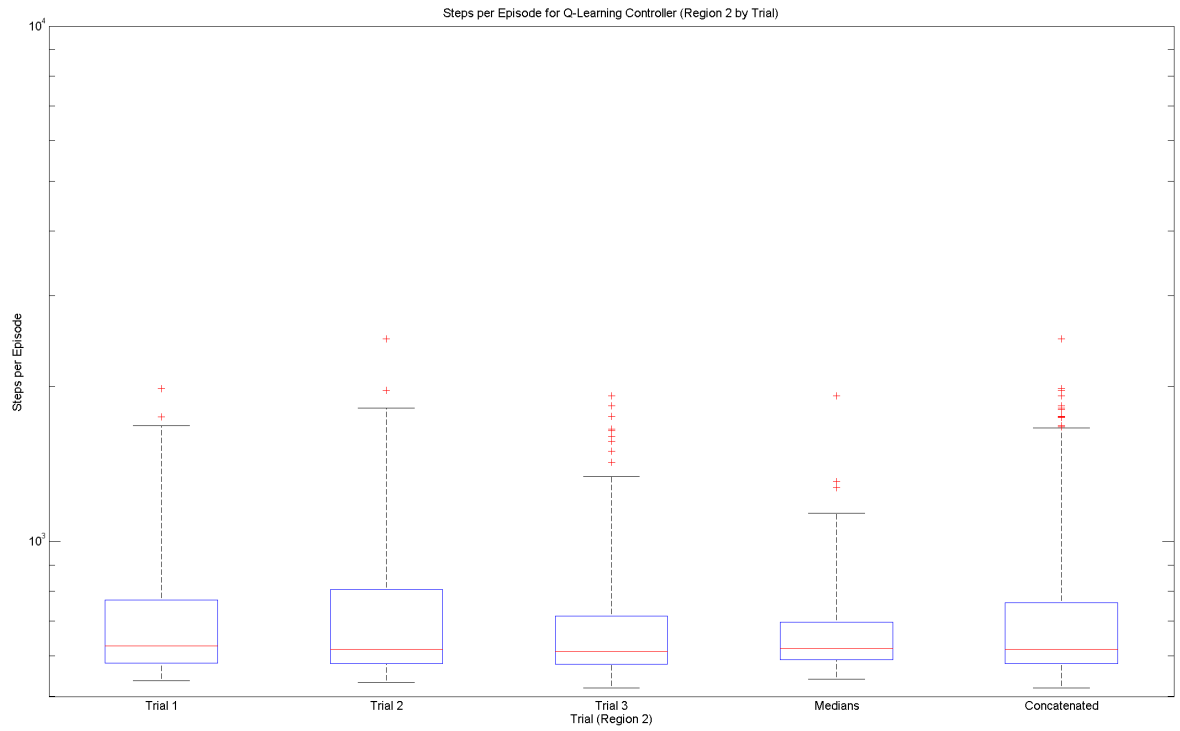


Figure 6.70: Distribution of episode lengths in Region 2 for QL controller (final baseline).

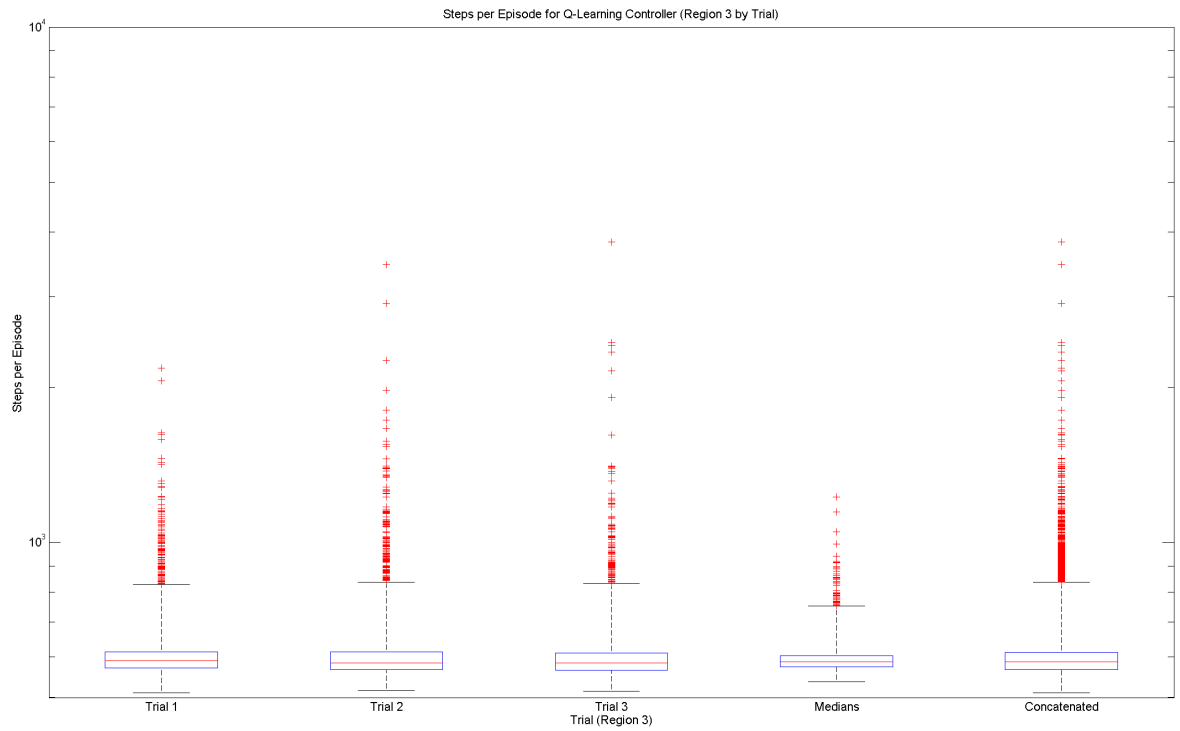


Figure 6.71: Distribution of episode lengths in Region 3 for QL controller (final baseline).

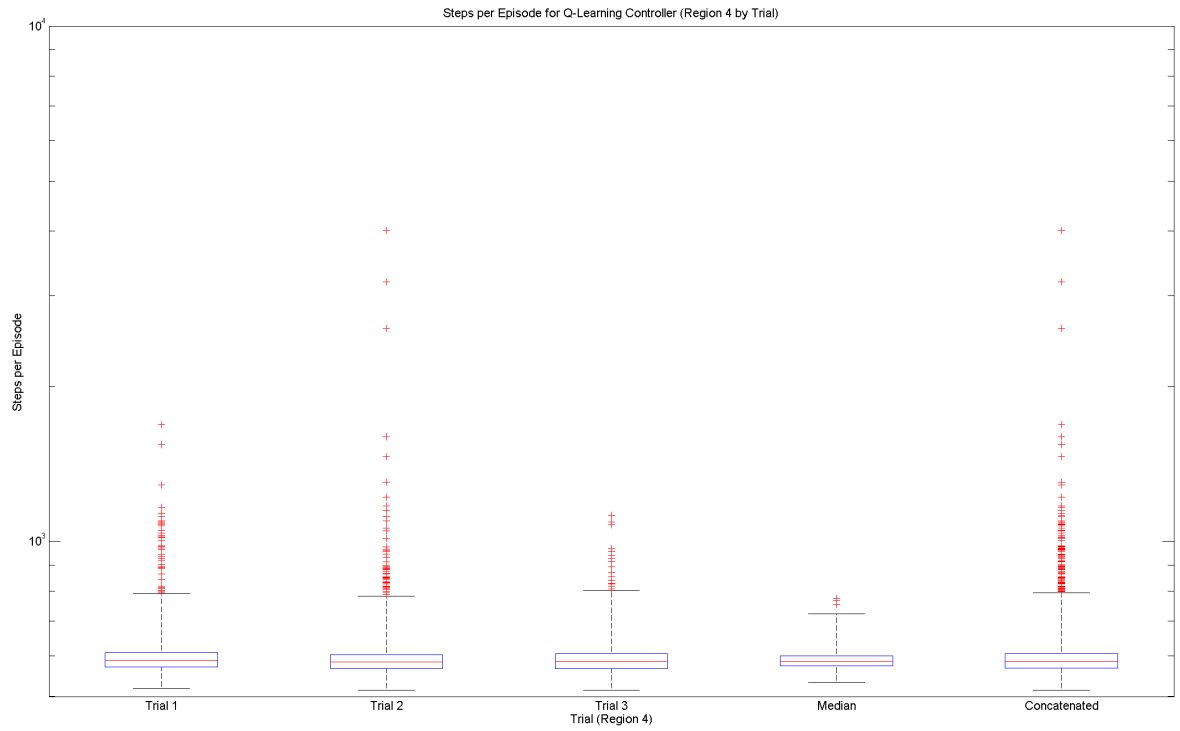


Figure 6.72: Distribution of episode lengths in Region 4 for QL controller (final baseline).

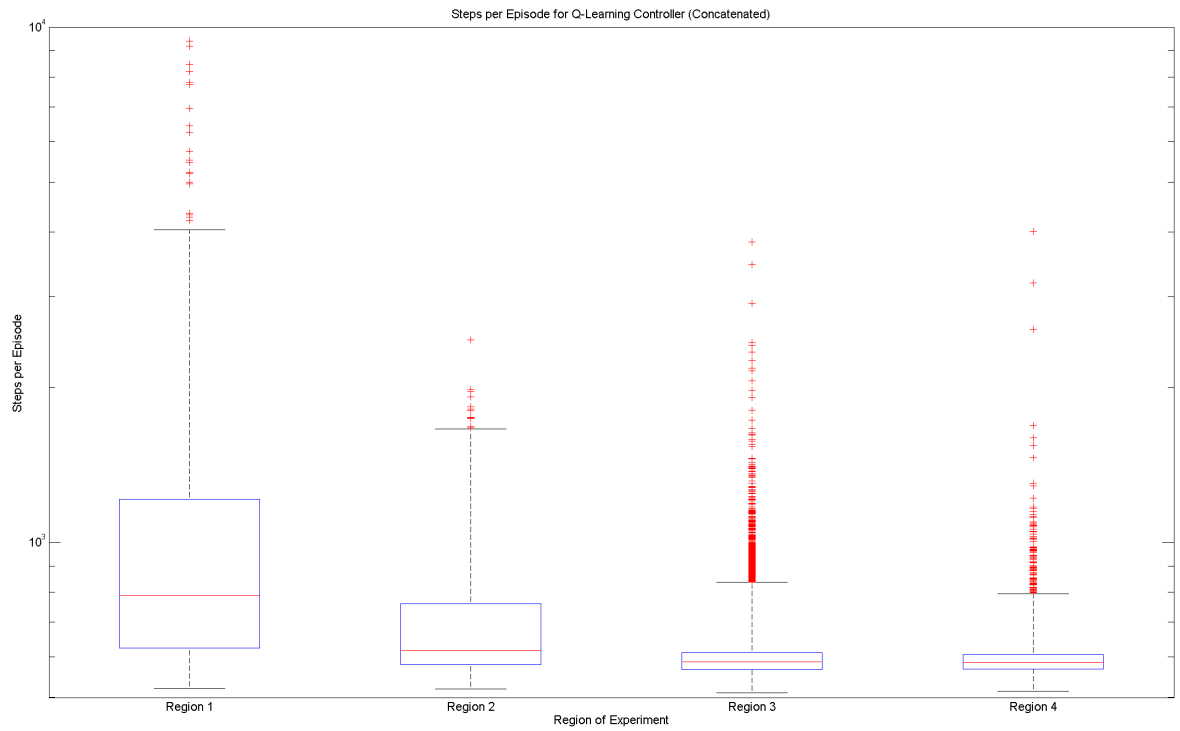


Figure 6.73: Distribution of concatenated episode lengths for QL controller (final baseline).

<b>Overall Performance</b>				
Total Length (Episodes)	7740			
Steady State Median Length (Steps)	587			
	Region 1	Region 2	Region 3	Region 4
Length (Episodes)	278	288	3587	3587
<b>Episode Length (Steps)</b>				
Mean	1516	724	613	594
Median	788	618	586	585
Standard Deviation	4395	248	118	70
Lower Quartile	624	579	567	568
Upper Quartile	1212	762	612	606
Minimum	521	519	510	514
Maximum	65826	2470	3828	4010

Table 6.43: KPIs for QL controller (final baseline).

Compared against the initial baseline experiment, the final baseline experiment is significantly superior in almost every performance characteristic: steady-state median episode length is only 90% of that initial length, plus the learner behaviour is much more stable: whilst the standard deviation in region one is very similar to the initial experiment, in region two the standard deviation falls to only 11% of that in the initial experiment, which contributes to the length of region two falling to only 65% that of the initial baseline. The difference in performance is clearly visible by comparing Figures 6.2 and 6.68.

Compared against the performance of the critically damped PID controller tested in §5.2.2, the steady-state performance of the Q-Learning based controller in its final configuration is particularly interesting: the median episode length for the Q-Learning controller almost identical (587 steps against 589 for the PID controller), but standard deviation remains considerably higher (70 against 31 for the PID controller). The upper quartile for the Q-Learning controller is located lower than for the PID controller (606 against 624): the statistical performance of the Q-Learning controller is adversely affected by the presence of outlier episodes with long durations. The maximum episode length in region four for the Q-Learning controller is 4010 (or 6.8 times the median episode length), whereas the maximum episode length for the PID controller is only 674 (or just 1.1 times the median episode length). If not for these outliers, then the performance of the Q-Learning controller would likely be superior to that of the PID controller. Additionally, as expected, the performance of the learning controller is much worse than the PID controller initially, which leads the PID controller to complete slightly more episodes in total, despite the similarity in median performance towards the end of the experiment (7740 episodes against 7960 for the PID controller).

Finally, consideration is also given to the time-domain response of the controller. A sample plot of the response is shown in Figure 6.74. The response of the controller remains similar to that of the  $\epsilon = 0.05$  case. The response of all three axes are underdamped; the damping is non-linear, so that there is substantial initial overshoot in the step response for large errors, but further oscillations are arrested immediately following this. The most significant difference in the time-domain response from that of the initial baseline is that the response in the yaw axis is significantly faster, due to the absence of the severe overdamping present in the initial baseline experiment.

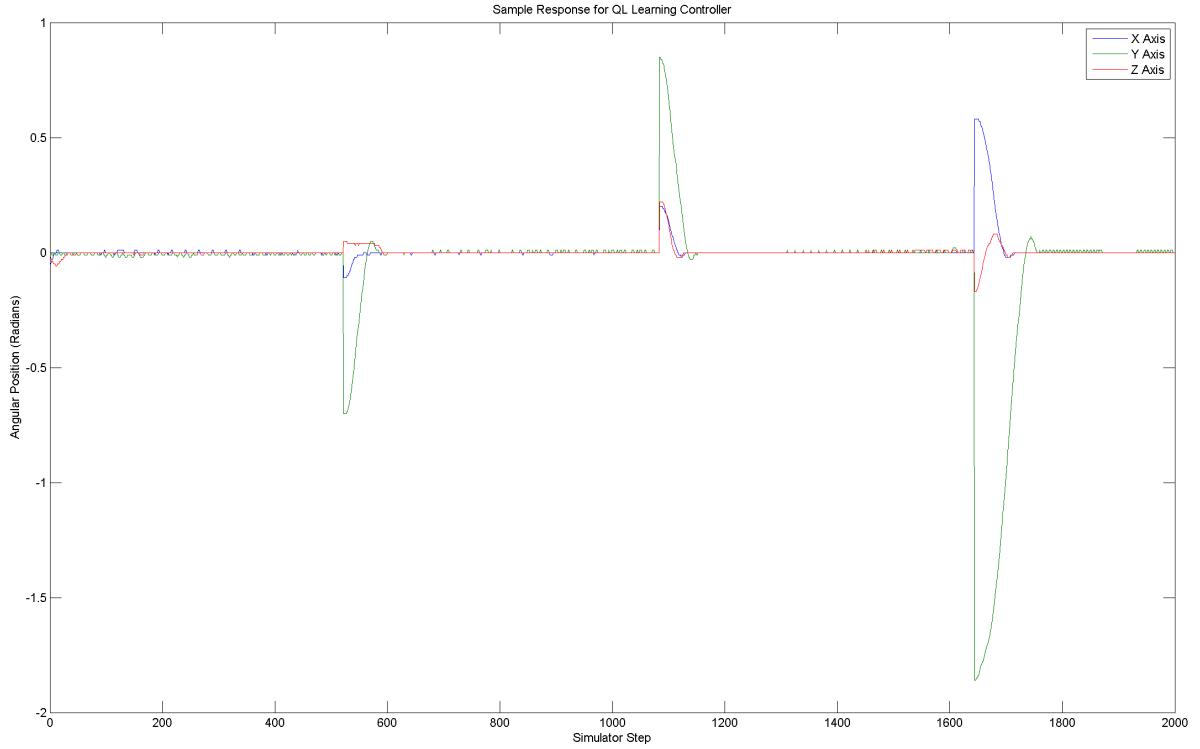


Figure 6.74: Sample time-domain response for QL controller (final baseline).

### 6.3 Effects of Disturbance

Because any practical control system must be able to handle disturbances applied to the plant, an additional experiment was performed to characterise the performance of the QL based controller in the presence of disturbances.

Disturbance was simulated in the same manner as used in §5.2.3: a randomly distributed moment was applied to the vehicle at each time-step. The disturbance component was independently normally distributed in each axis, with a mean of zero, and a standard deviation of  $2.5Nm^{-1}$ .

A set of three trials was simulated, using the parameter configuration from the final baseline experiment. The resulting episode lengths for each trial are plotted in Figure 6.75. Then, an identical method for collating and analysing the results from each trial was followed as used in §6.2.1. The resulting smoothed set of episode lengths is shown in Figure 6.76.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.65. The key performance characteristics derived from the experiment are also shown in Table 6.41.

Compared against the performance of the final baseline experiment, there is degradation of the controller performance in terms of episode length: an increase of around 8%. Additionally, there is an increase in required time for the learner to settle, as indicated by an increase in the lengths of regions one and two. This results in a reduction of around 15% for total number of episodes completed. However, this compares favourably against the reduction in performance of the PID controller (as detailed in §5.2.3) when exposed to disturbances: a reduction in total number of



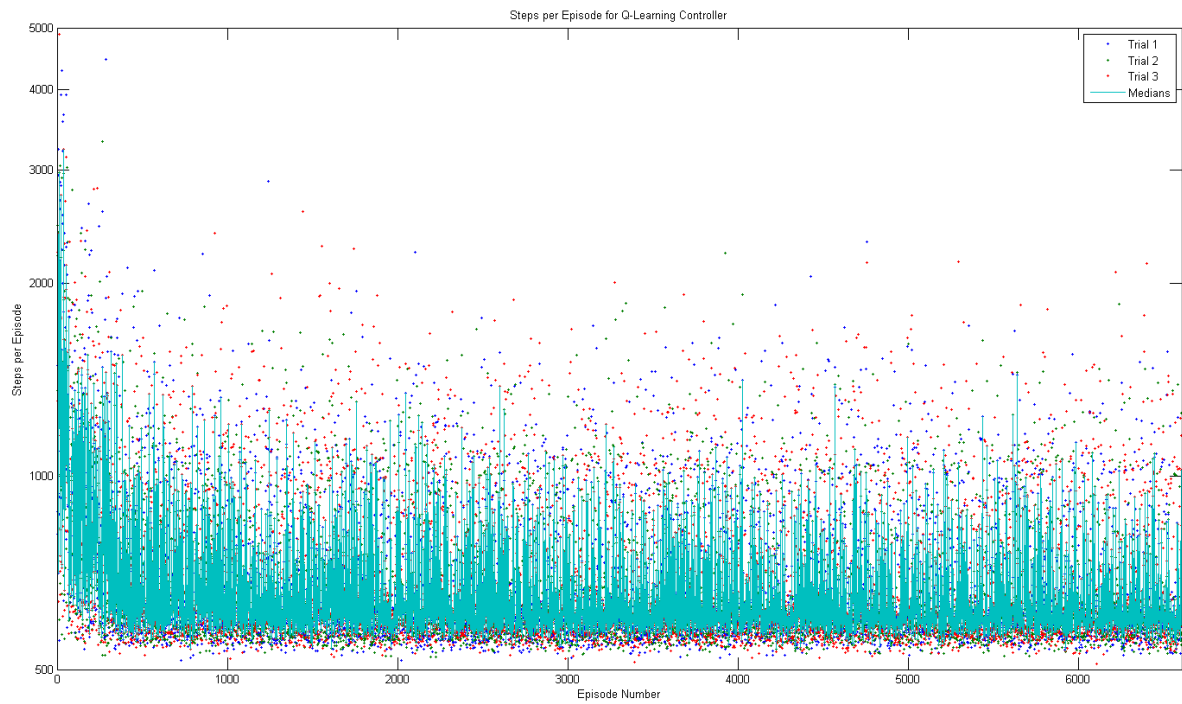


Figure 6.75: Individual episode lengths for QL controller with disturbances.

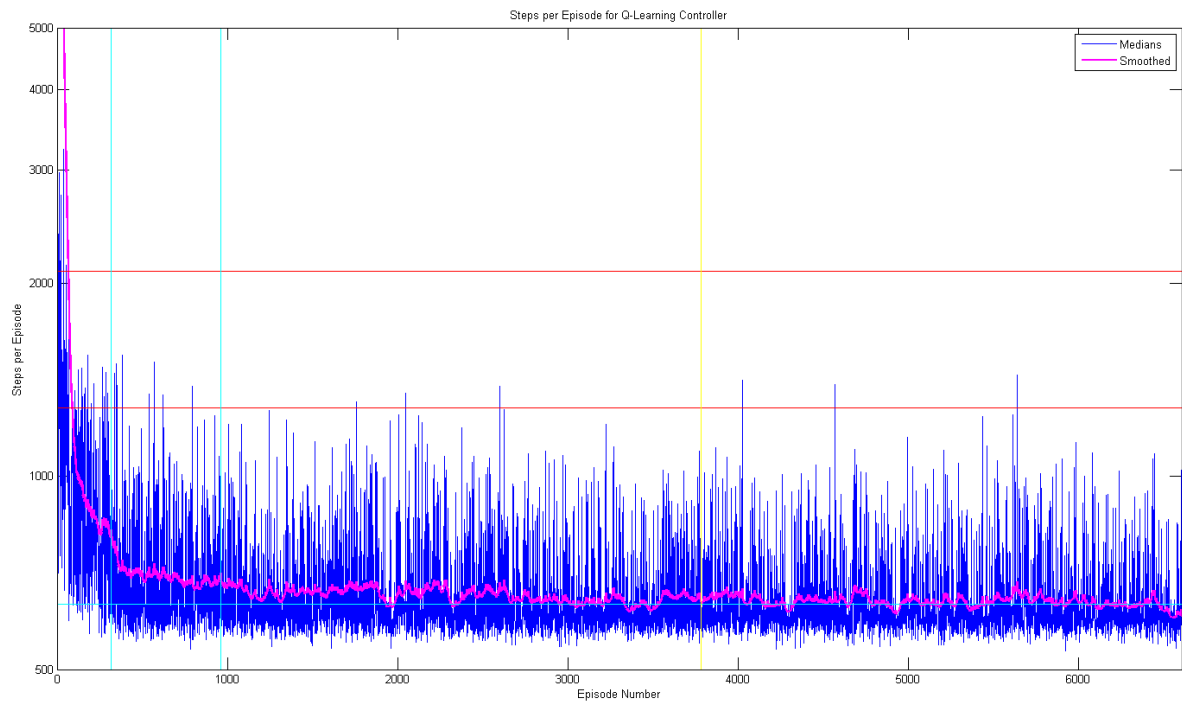


Figure 6.76: Smoothed episode lengths for QL controller with disturbances.

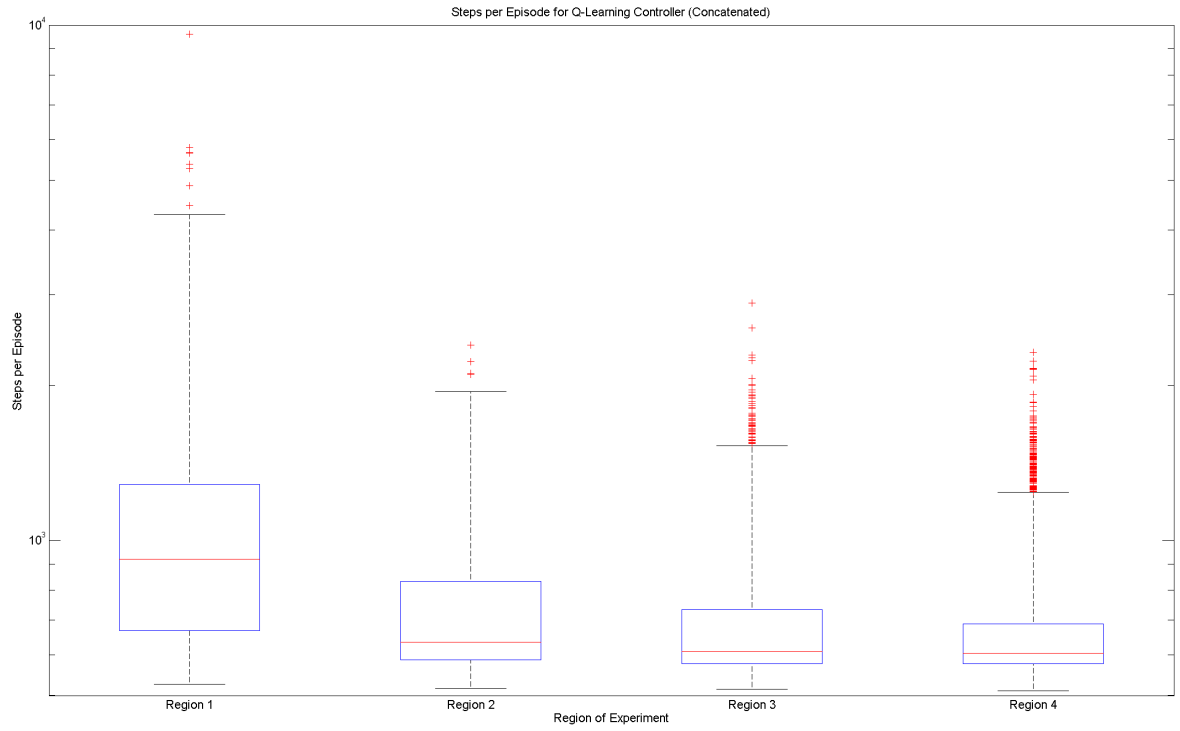


Figure 6.77: Distribution of concatenated episode lengths for QL controller with disturbances.

<b>Overall Performance</b>				
Total Length (Episodes)	6606			
Steady State Median Length (Steps)	632			
	Region 1	Region 2	Region 3	Region 4
Length (Episodes)	318	642	2823	2823
<b>Episode Length (Steps)</b>				
Mean	1389	746	697	680
Median	919	635	608	604
Standard Deviation	3590	248	206	188
Lower Quartile	669	586	577	577
Upper Quartile	1283	834	735	688
Minimum	526	516	514	510
Maximum	58359	2396	2885	2320

Table 6.44: KPIs for QL controller with disturbances.

episodes from 7960 to 3003 (or 62%). Compared against the PID controller, the performance of the Q-Learning based controller is now substantially better: median episode length is only 632 compared with 1646 (only 38%). In fact, the upper quartile value for the Q-Learning controller in regions three and four (735 and 688) are less than the lower quartile value for the PID controller (798).

The reason for this upset in performance is that whilst the PID controller is highly susceptible to noise in the input signal, due to the unfiltered derivative action, the Q-Learning controller appears to be affected only minimally. Part of this immunity to noise created by the disturbances may be due to the limited resolution of the velocity space which the Q-Function estimator uses (the parameter  $\sigma_{\dot{\theta}}$ ).

Consideration of a sample of the time-domain response of the controller, as shown in Figure 6.78, shows that the control response settles to be fairly close to the nominal position in around 100 simulator steps, which is not substantially different to the response seen in the undisturbed final baseline case. However, the presence of disturbance means that it can take substantially longer for the controller to achieve the required number of sequential steps without the error growing larger than the threshold value, even if momentarily.

## 6.4 Handling Changes in Dynamics

As discussed in §1.1, dealing with variations in plant dynamics is a significant issue in control system design; it is hoped that use of RL based algorithms could be beneficial in this regard. To assess any such benefit, a further set of experiments was performed to characterise the performance of the Q-Learning based controller in the case where the plant dynamics are modified at run-time.

In these experiment, the first half of each trial (i.e. the first two and a half million steps) was conducted using the same simulator configuration as used previously. At the mid-point of each trial, the plant dynamics were adjusted by increasing the mass of the vehicle by a factor of three (from  $m = 10$  to  $m = 30$ ). The second half of the trial was then performed using the original controller instance, operating on the newly adjusted dynamics.

Two experiments of three trials each were performed, to characterise control performance both with and without the presence of disturbances.

### 6.4.1 Without Disturbances

A set of three trials was simulated, using the parameter configuration from the final baseline experiment. The resulting episode lengths for each trial are plotted in Figure 6.79. Because the change in dynamics during each trial is configured to occur after a fixed number of simulator steps, the number of episodes which occur prior to the dynamics changing will be slightly different in each trial. This means that the method of collating and analysing the results from each trial used previously (as detailed in §6.2.1), will not be suitable here.

Instead of aggregating the three trials by selecting the median, the series of episode lengths during each trial is smoothed separately (using the same moving average mechanism used previously), and divided into regions (using the same method used previously) separately. The smoothed set of episode lengths, along with the region boundaries is shown in Figure

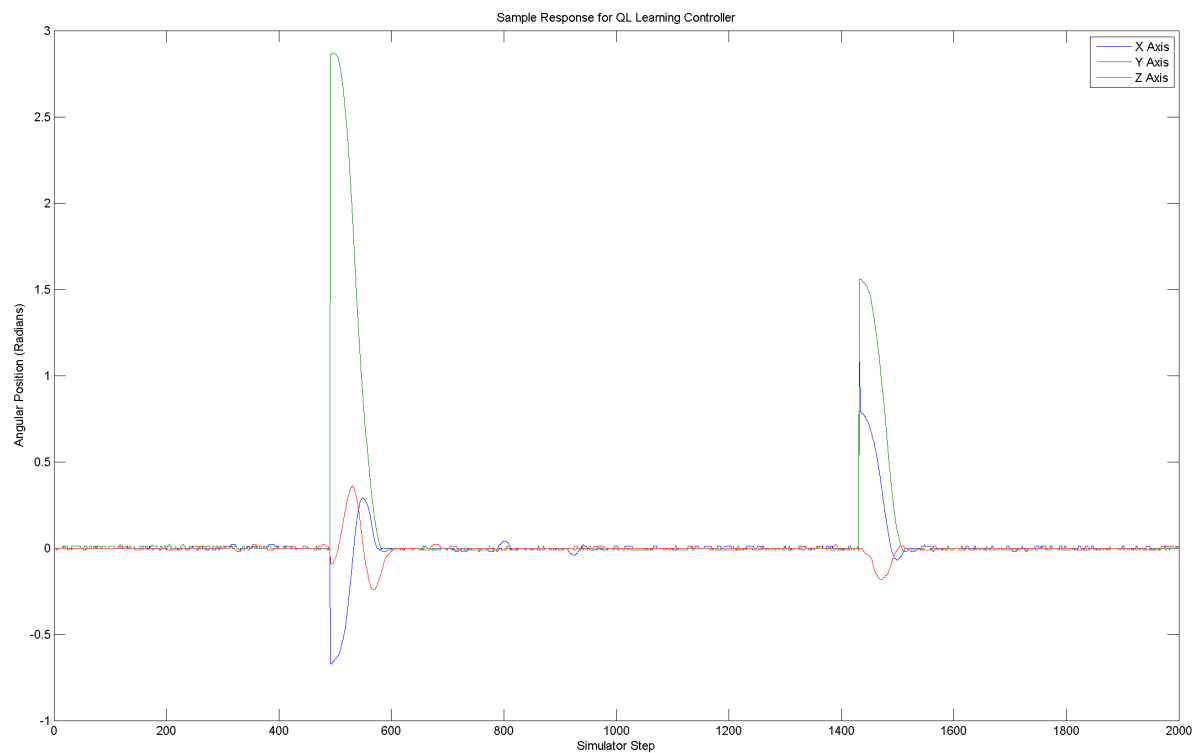


Figure 6.78: Sample time-domain response for QL controller with disturbances.

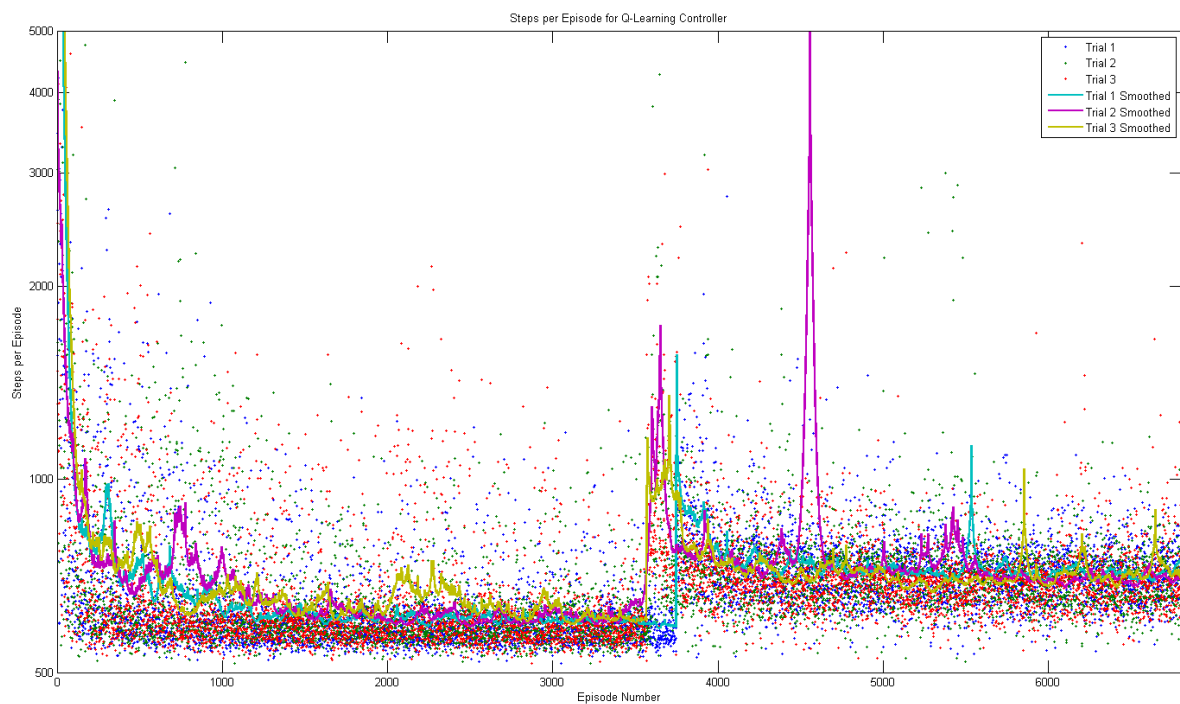


Figure 6.79: Individual episode lengths for QL controller with modified plant dynamics.

6.80. The horizontal lines (indicating the median steady-state episode length) and vertical lines (indicating the borders between analysis ‘regions’) are colour coded to match the trial they are associated with. Each trial is divided into two portions; before and after the change in plant dynamics.

Once the trials have been split into regions, the series data from each region is concatenated across the three trials. This creates a single set of concatenated data which can continue to be analysed in the same manner as used in previous experiments. The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.81 and Figure 6.81 (before and after the change in plant dynamics, respectively). The key performance characteristics derived from the experiment are also shown in Table 6.45 and Table 6.46 (before and after the change in plant dynamics, respectively).

As expected, the performance prior to the dynamics change is nominally similar to that of the final baseline experiment. There is some variation in regions one and two, but this is mostly caused by the alternate smoothing mechanism used in this experiment, which results in regions one and two being reported as longer than when using the smoothing method used for previous experiments. As identified in the final baseline experiment, the steady-state median performance of the Q-Learning controller is fairly similar to that of the PID controller, but the standard deviation remains substantially higher (around 80, compared to 30 for the PID controller).

Immediately after the change in dynamics, the performance of the Q-Learning controller degrades, but then flattens towards a steady-state again. The new steady-state median remains higher than that prior to the change in plant dynamics, approximately equal to the median performance of the equivalent PID controller. However, the standard deviation of episode length remains around 80. Since in the PID controller case, the standard deviation of episode lengths increases when the plant dynamics are changed, this leaves the Q-Learning controller with a standard deviation of episode lengths in fact slightly better than the PID controller.

Finally, consideration is also given to the time-domain response of the controller, before and after the change in dynamics. A sample plot of each response is shown in Figure 6.83 and Figure 6.84. As expected, the response before the plant dynamics change is nominally the same as in the final baseline experiment. After the plant dynamics change, there is more pronounced underdamping across all three axes. This is consistent with the change in dynamics; the plant mass has increased, so will require more damping to prevent overshoot.

<b>Overall Performance</b>				
Total Length (Episodes)	3440			
Steady State Median Length (Steps)	609			
	<b>Region 1</b>	<b>Region 2</b>	<b>Region 3</b>	<b>Region 4</b>
Length (Episodes)	684	813	972	971
<b>Episode Length (Steps)</b>				
Mean	951	648	612	607
Median	654	600	587	588
Standard Deviation	2057	155	87	79
Lower Quartile	592	574	567	566
Upper Quartile	923	638	618	615
Minimum	521	516	519	516
Maximum	68294	2145	1390	1283

Table 6.45: KPIs for QL controller, before modifying plant dynamics.

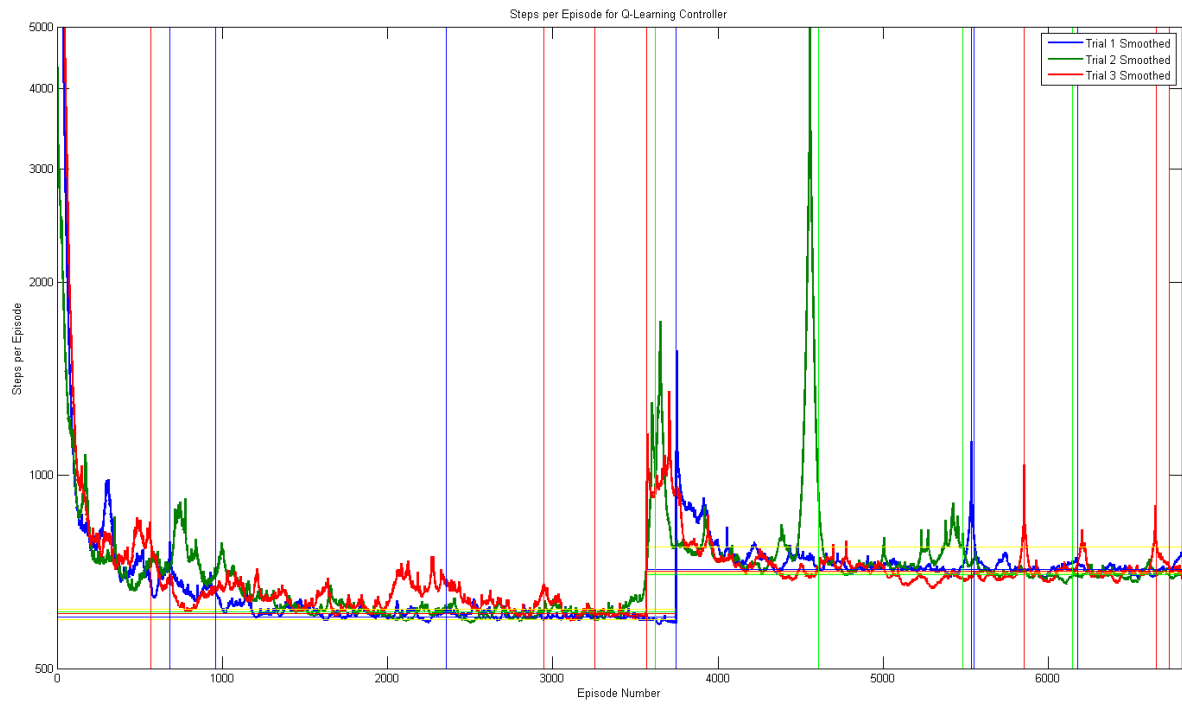


Figure 6.80: Smoothed episode lengths for QL controller with modified plant dynamics.

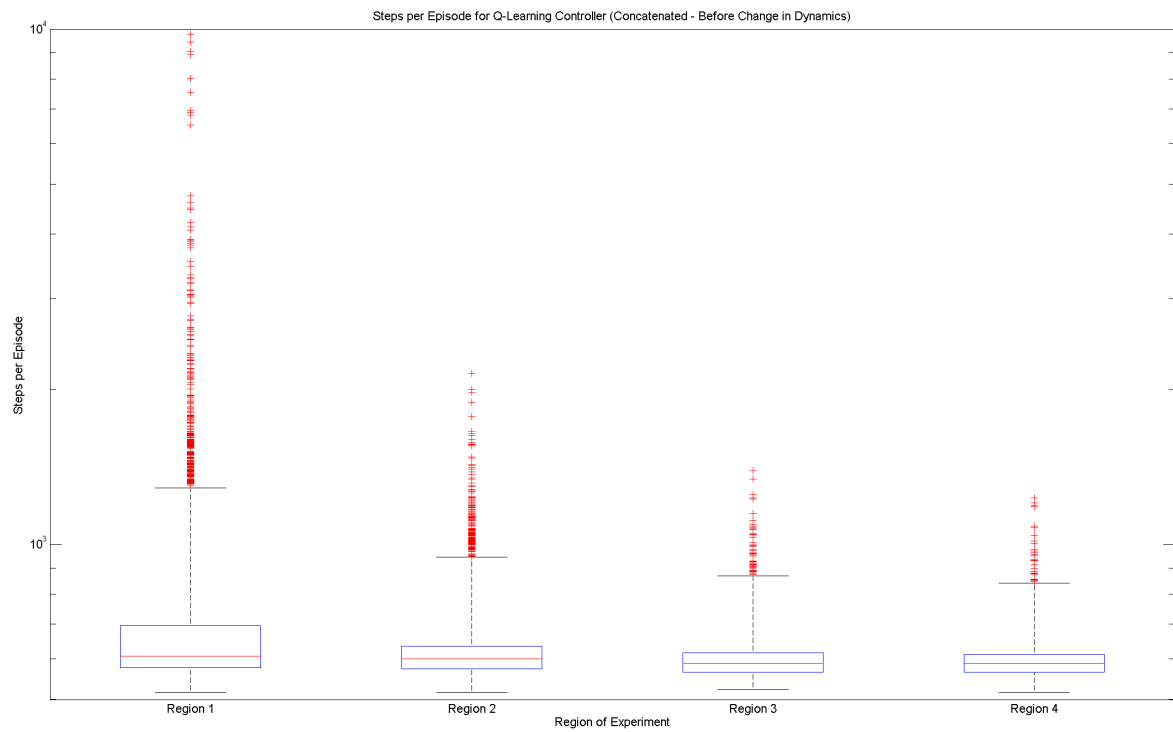


Figure 6.81: Distribution of concatenated episode lengths for QL controller, before modifying plant dynamics.

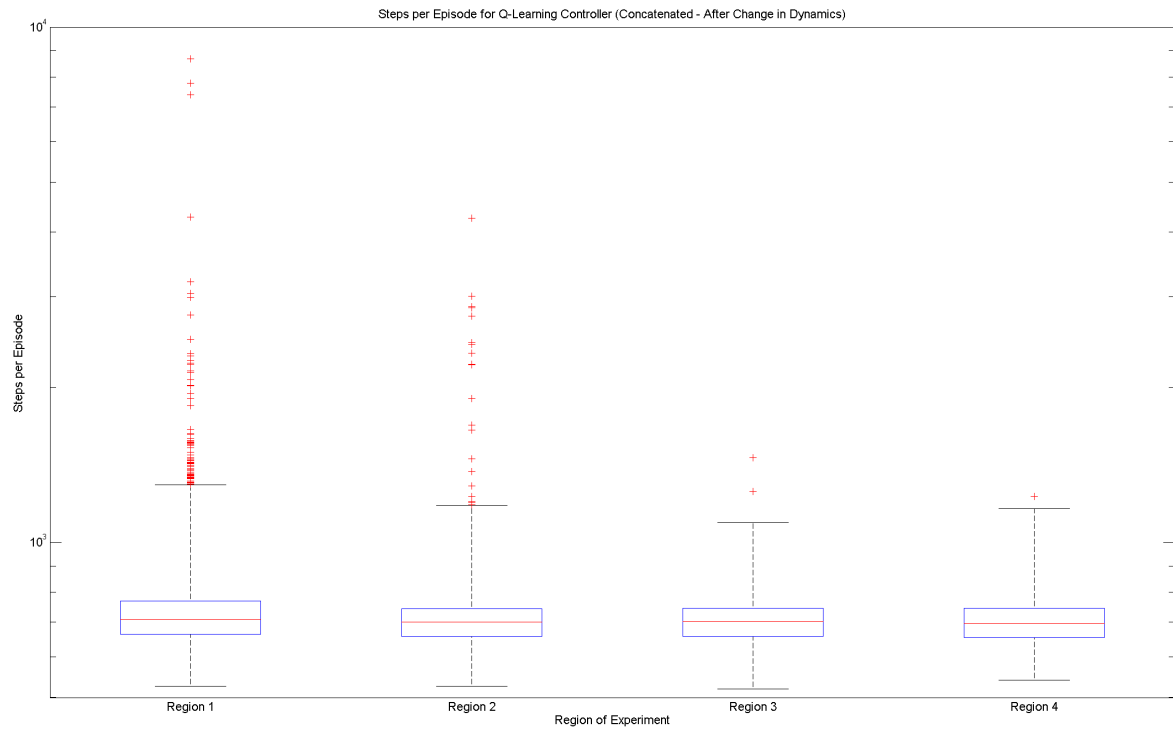


Figure 6.82: Distribution of concatenated episode lengths for QL controller, after modifying plant dynamics.

<b>Overall Performance</b>				
Total Length (Episodes)	3842			
Steady State Median Length (Steps)	709			
	Region 1	Region 2	Region 3	Region 4
Length (Episodes)	1787	799	628	628
<b>Episode Length (Steps)</b>				
Mean	782	724	710	707
Median	710	700	703	696
Standard Deviation	1709	188	79	76
Lower Quartile	664	657	657	654
Upper Quartile	770	744	745	745
Minimum	526	526	520	540
Maximum	118556	4257	1459	1227

Table 6.46: KPIs for QL controller, after modifying plant dynamics.

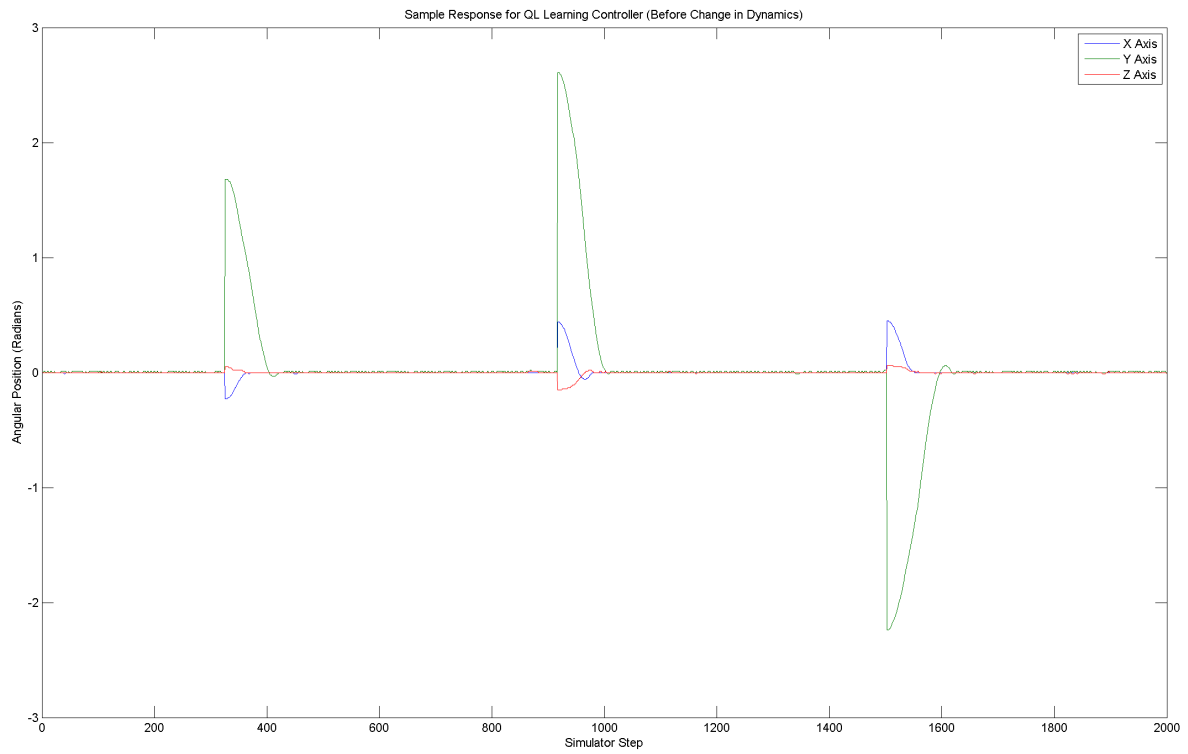


Figure 6.83: Sample time-domain response for QL controller, before modifying plant dynamics.

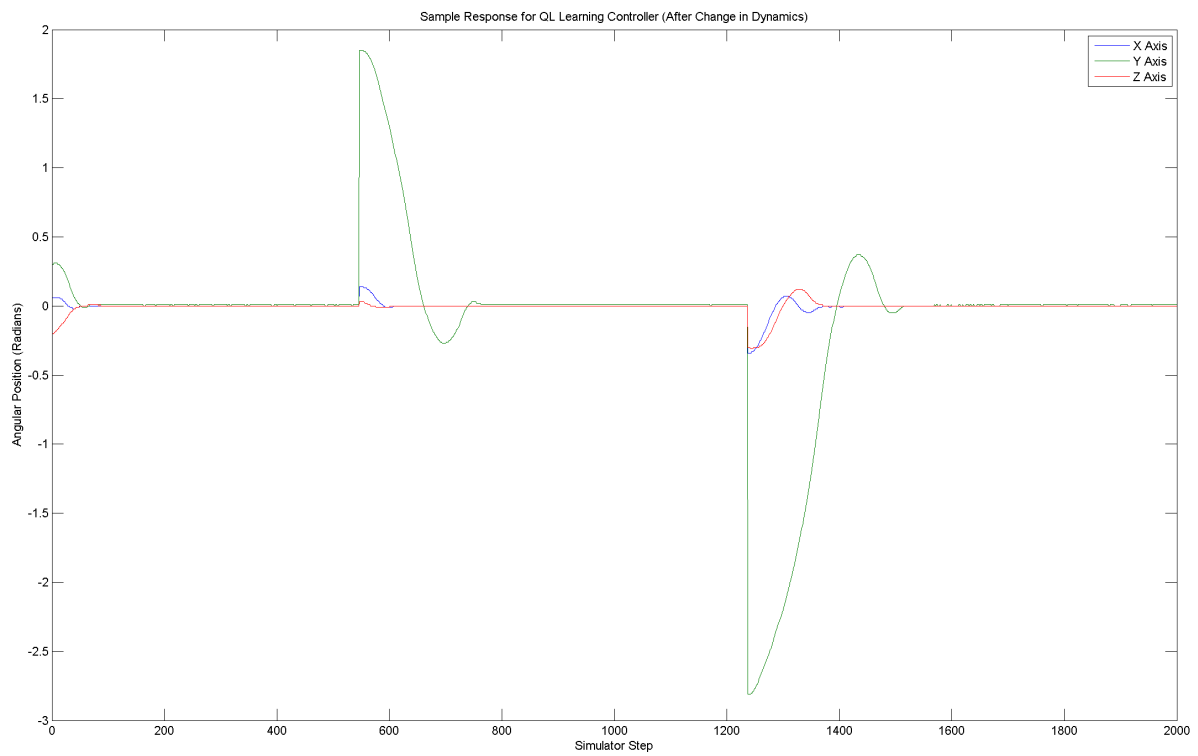


Figure 6.84: Sample time-domain response for QL controller, after modifying plant dynamics.



### 6.4.2 With Disturbances

A set of three trials was simulated, using the parameter configuration from the final baseline experiment. The resulting episode lengths for each trial are plotted in Figure 6.85. An identical method of collating and analysing the results of each trial is used as in the previous experiment. The smoothed set of episode lengths, along with the region boundaries is shown in Figure 6.86.

The distribution of episode lengths over the complete experiment, based upon the concatenated results within each region, is shown in Figure 6.87 and Figure 6.88 (before and after the change in plant dynamics, respectively). The key performance characteristics derived from the experiment are also shown in Table 6.47 and Table 6.48 (before and after the change in plant dynamics, respectively).

As expected, the performance prior to the dynamics change is similar to that of the final baseline experiment with disturbances. The median episode lengths are slightly longer, and the standard deviation of episode lengths is greater. This is assumed to be a function of the stochastic behaviour of the disturbances (since only a few unusually long episodes can have a significant affect on the distribution of episode lengths), and also because this part of the experiment is only half the length of the full experiment performed in §6.3, which gives the learner less time to settle. As in the baseline experiment with disturbances, the performance of the Q-Learning controller is superior to that of the PID Controller.

Immediately after the change in dynamics, the performance of the Q-Learning controller degrades, but then begins to flatten towards a steady-state again. The experiment ends before the learner has opportunity to fully reach steady-state. At the end of the experiment, the median episode length is higher than that of the equivalent PID Controller (at 1246 steps per episode, against 1075 for the PID controller). The standard deviation is also higher (at 884, against 771 for the PID controller). A longer experiment would be necessary to determine how these performance metrics compared once the learner had fully stabilised.

Finally, consideration is also given to the time-domain response of the controller, before and after the change in dynamics. A sample plot of each response is shown in Figure 6.89 and Figure 6.90. As expected, the response before the plant dynamics change is nominally the same as in the disturbed final baseline experiment; increased settling time due to disturbances. After the plant dynamics change, there is more pronounced underdamping across all three axes.

<b>Overall Performance</b>				
Total Length (Episodes)	2672			
Steady State Median Length (Steps)	811			
	<b>Region 1</b>	<b>Region 2</b>	<b>Region 3</b>	<b>Region 4</b>
Length (Episodes)	349	2136	94	93
<b>Episode Length (Steps)</b>				
Mean	1318	841	809	834
Median	983	691	638	639
Standard Deviation	2666	357	311	334
Lower Quartile	687	594	588	591
Upper Quartile	1375	982	944	955
Minimum	532	504	539	535
Maximum	55843	4954	2220	2119

Table 6.47: KPIs for QL controller with disturbances, before modifying plant dynamics.

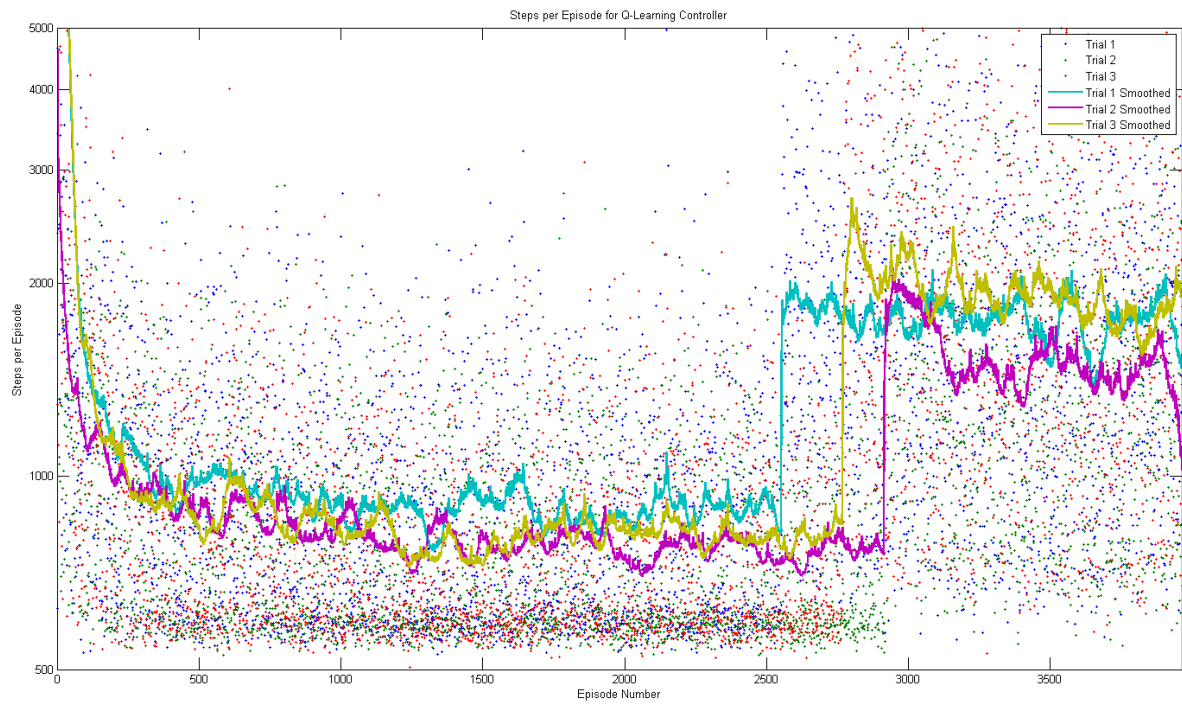


Figure 6.85: Individual episode lengths for QL controller with disturbances and modified plant dynamics.

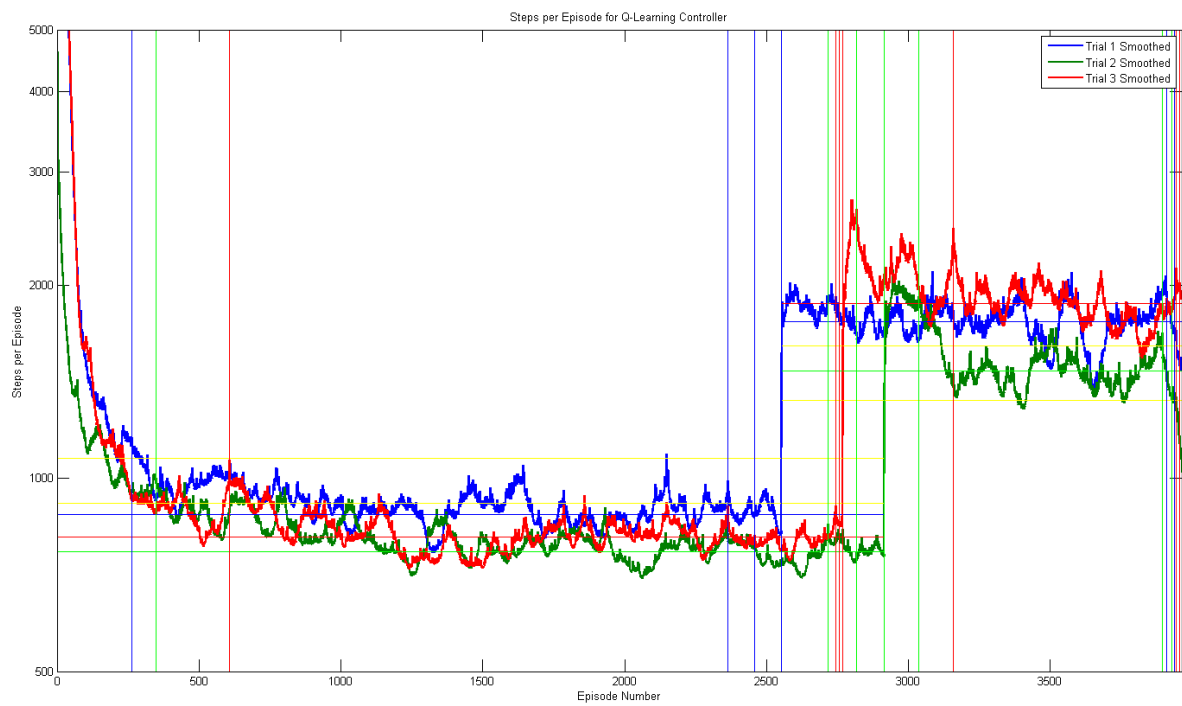


Figure 6.86: Smoothed episode lengths for QL controller with disturbances and modified plant dynamics.

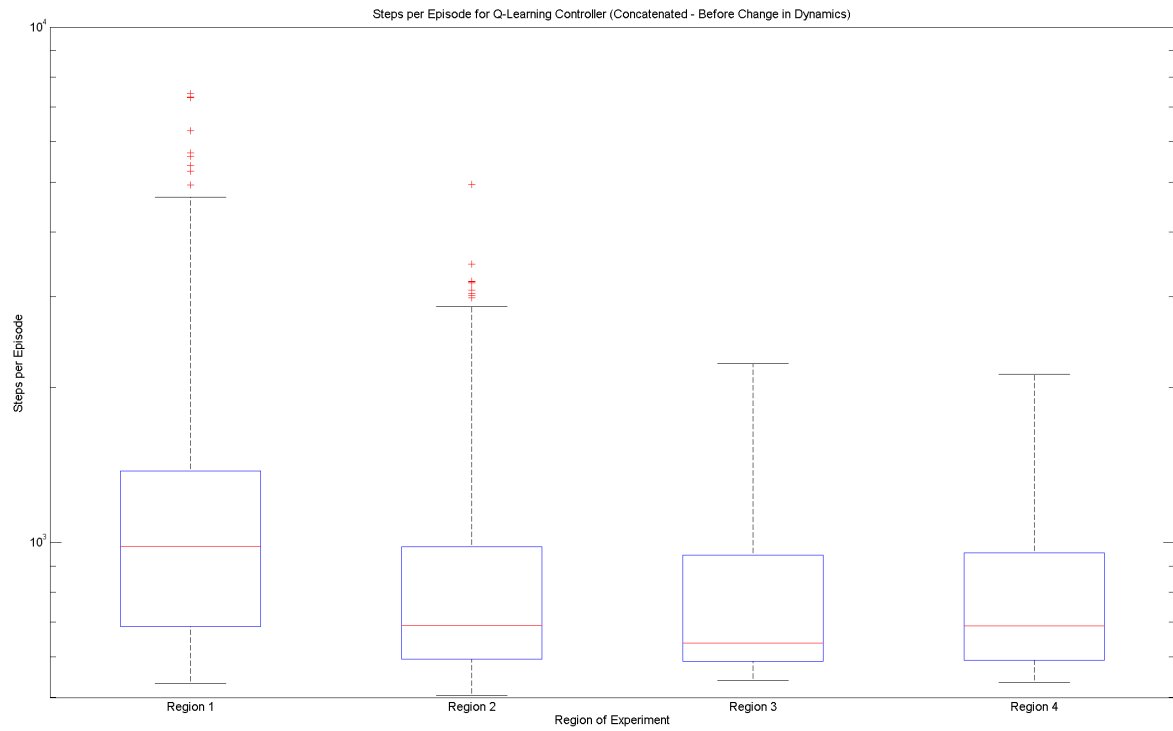


Figure 6.87: Distribution of concatenated episode lengths for QL controller with disturbances, before modifying plant dynamics.

<b>Overall Performance</b>				
Total Length (Episodes)	1033			
Steady State Median Length (Steps)	1754			
	Region 1	Region 2	Region 3	Region 4
Length (Episodes)	123	856	27	27
<b>Episode Length (Steps)</b>				
Mean	2097	1714	1525	1533
Median	1687	1405	1290	1246
Standard Deviation	1476	1067	736	884
Lower Quartile	1084	944	955	841
Upper Quartile	2666	2153	2007	2026
Minimum	532	530	576	590
Maximum	12310	9771	3645	3937

Table 6.48: KPIs for QL controller with disturbances, after modifying plant dynamics.

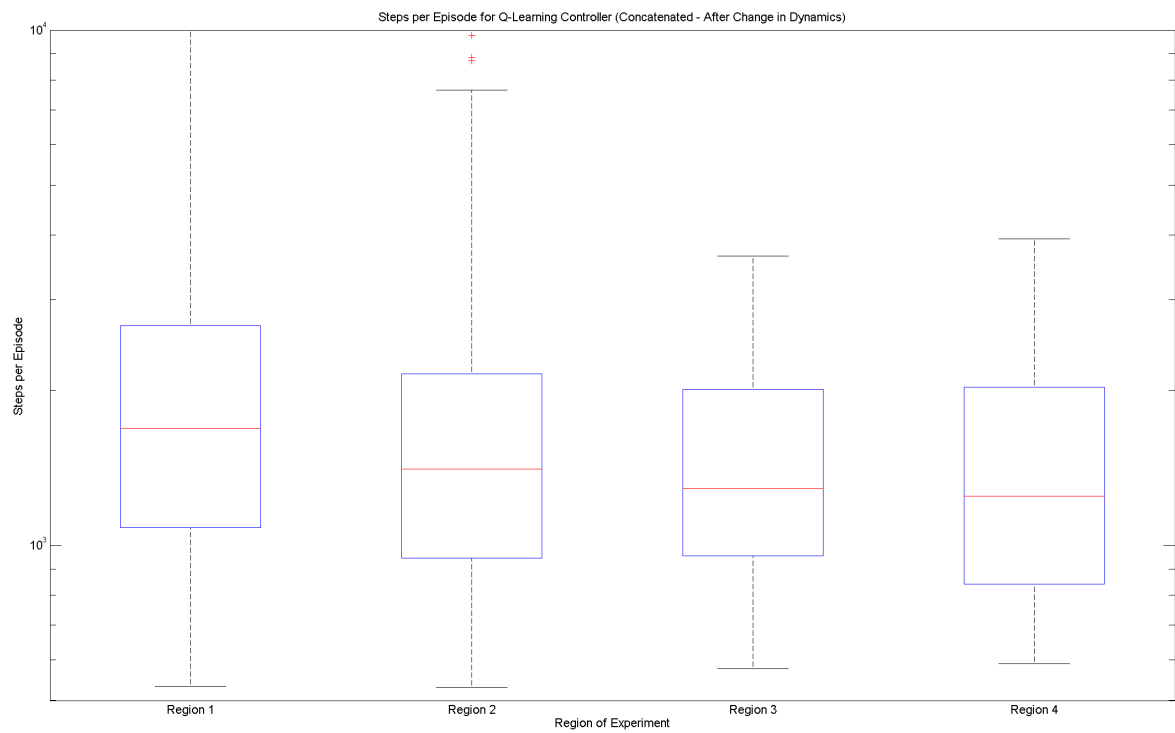


Figure 6.88: Distribution of concatenated episode lengths for QL controller with disturbances, after modifying plant dynamics.

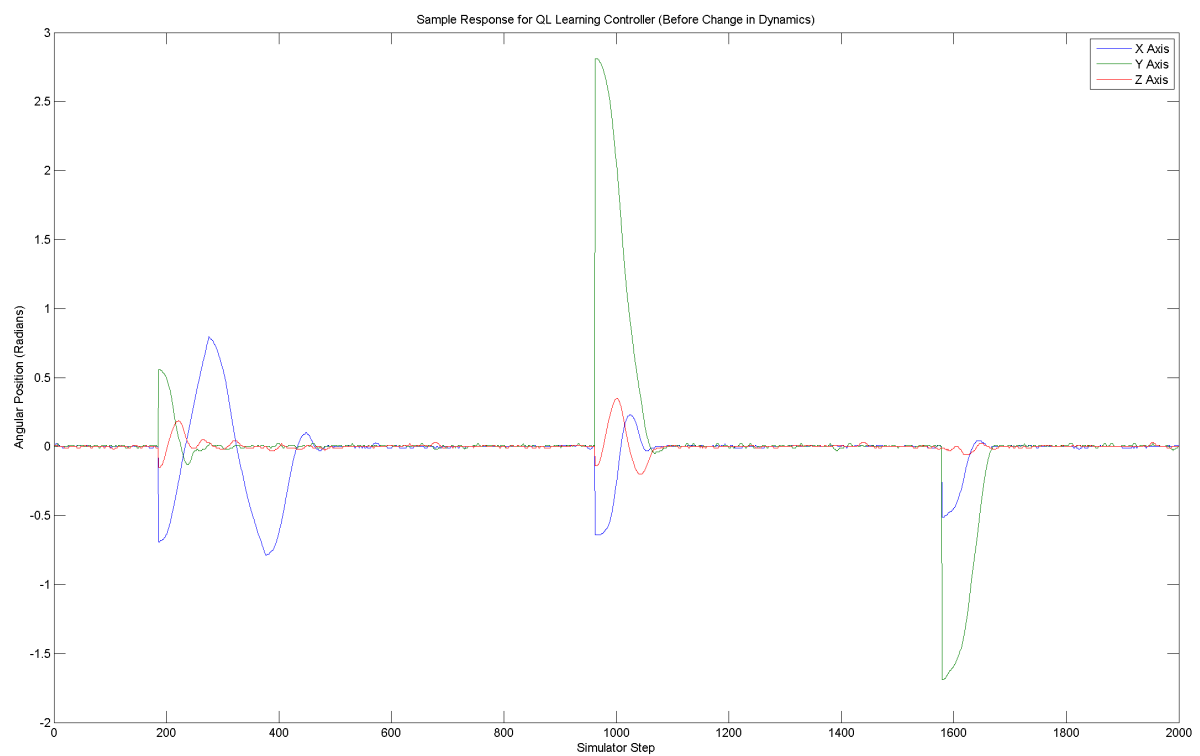


Figure 6.89: Sample time-domain response for QL controller with disturbances, before modifying plant dynamics.

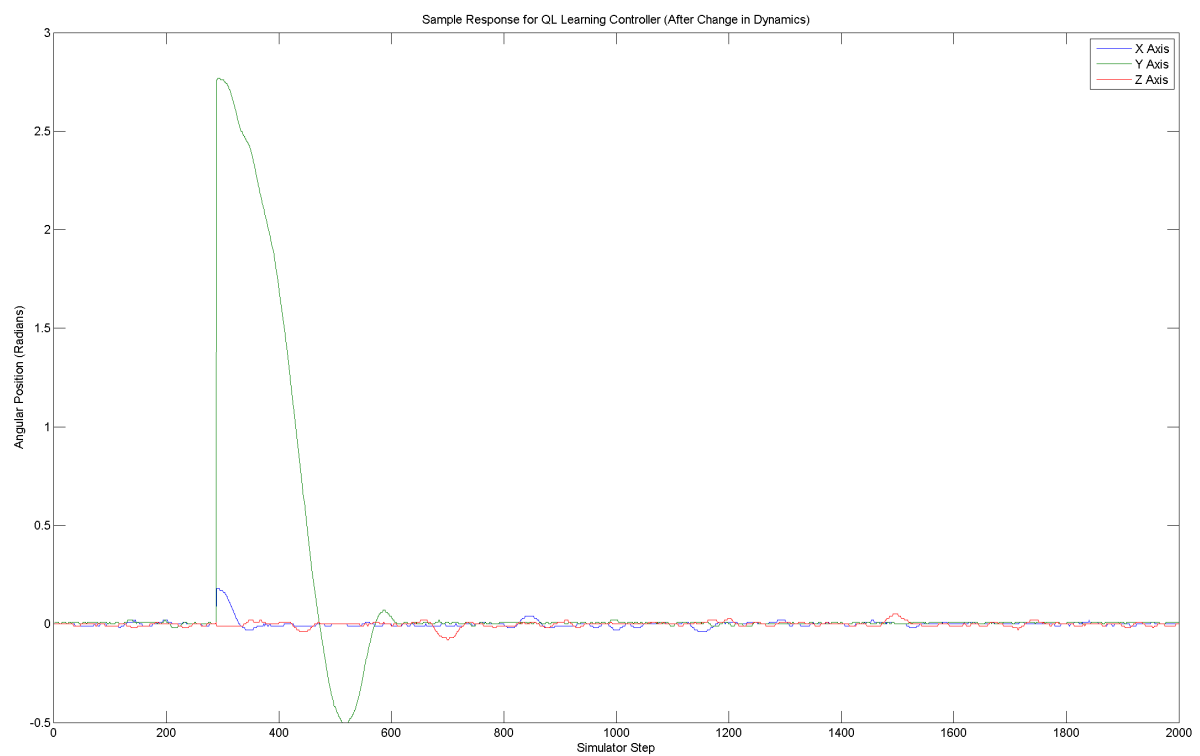


Figure 6.90: Sample time-domain response for QL controller with disturbances, after modifying plant dynamics.

## 6.5 Inspection of Q-Functions and Policies

During each experiment, the software can be configured to produce plots of the current Q-function and policy, to allow visualisation of how they change across the state-space of the environment.

Note that a Q-function maps Q-values to state/action pairs; since there are multiple actions, at each location in state-space, there are actually multiple Q-values. To be more correct, the software plots the *maximum* of the Q-values associated with each point in state-space.

A typical Q-function, as learned after a complete experimental trial (i.e. five million steps), is shown as Figure 6.91. The corresponding policy is shown as Figure 6.92.

The Q-function clearly shows a number of characteristics which are consistent with what would be expected: the Q-function yields a maxima at the nominal position (where the immediate reward values are zero); the Q-function tapers off to a minima near the edges of the state-space (where the immediate reward is negative). This variation in reward is what fundamentally drives the robot towards the nominal position. Note the ‘ridge’ which forms in the centre is angled from vertical, this is because the reward associated with having a velocity *towards* the nominal position is significantly different to that associated with a velocity *away* from the nominal position. The ‘high value’ areas in the top-left and bottom-right corners of the plot represent unexplored areas of the state-space, which remain at the high default value<sup>1</sup> because it is not physically possible for the robot to reach these areas: for example, the situation where the quadrotor is positioned against the limits of the roll axis, but has *maximum* velocity *away* from the limit, cannot be reached, because the acceleration in each axis is finite.

The policy plot is fairly noisy, but it also shows characteristics consistent with what would be expected: one side of the ‘ridge’ visible on the Q-function is predominantly associated with one action, whilst the other side is predominantly associated with the other. There is still considerable ‘noise’ in the policy, especially in areas which are traversed infrequently; this would be expected to decrease with further learning.

A second set of typical plots (from a different trial using the same experimental configuration) is shown as Figure 6.93 and Figure 6.94. Inspection of these plots shows that they feature largely the same characteristics as for the previous set of plots.

The ability to identify characteristics in the Q-function and policy plots which correlate with expected behaviour is useful, because it gives the developer confidence that the reinforcement learning algorithm is working correctly. However, the plots are difficult to use for any more complicated analysis, because very small changes in Q-function and policy in important areas of the state-space may result in significant variations in performance, and these small changes are difficult to identify visually. Consider the plots shown as Figure 6.95 and 6.96, which are taken from an experiment with significantly different parameters to the two shown previously. Whilst there are some visible differences, it is difficult to assess how these differences might affect control performance using just casual inspection.

Due to the visual similarity of Q-function and policy plots, even between trials with substantial differences in terms of control performance, direct analysis of Q-function and policy was not used in this project, beyond initially confirming that the algorithm was working correctly.

---

<sup>1</sup>A high default value is used because this drives the agent to explore any areas which remain unexplored, in the hope there may be ‘something of interest’ in these regions



Figure 6.91: A typical Q-Function for the X (roll) axis, after 5 million steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.

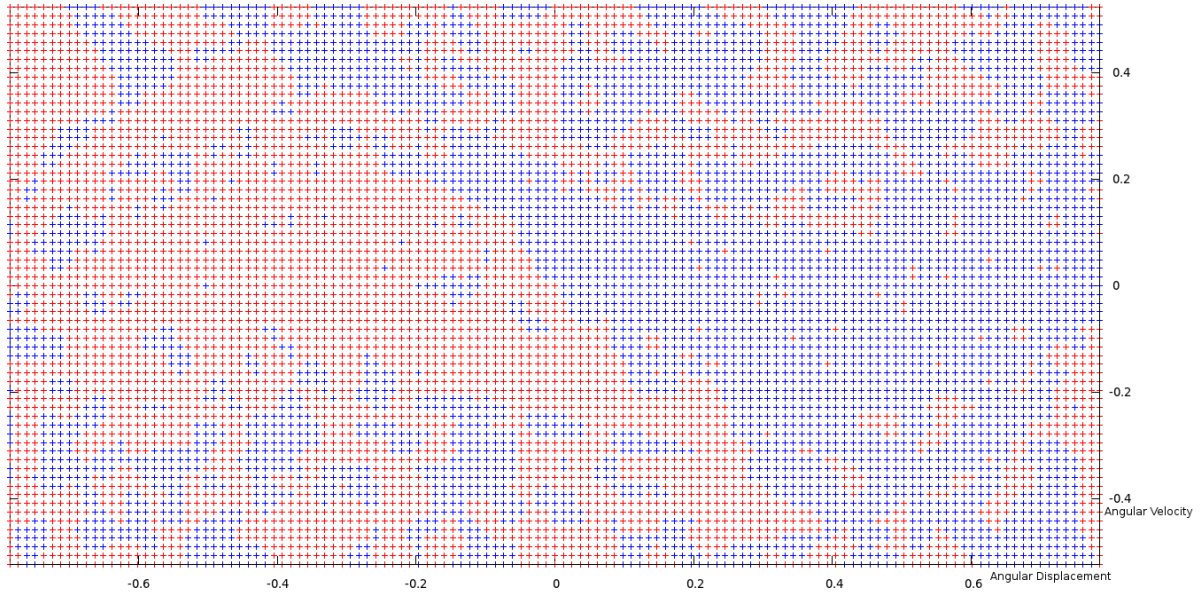


Figure 6.92: A typical policy function for the X (roll) axis, after 5 million steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.





Figure 6.93: Another typical Q-Function for the X (roll) axis, after 5 million steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.

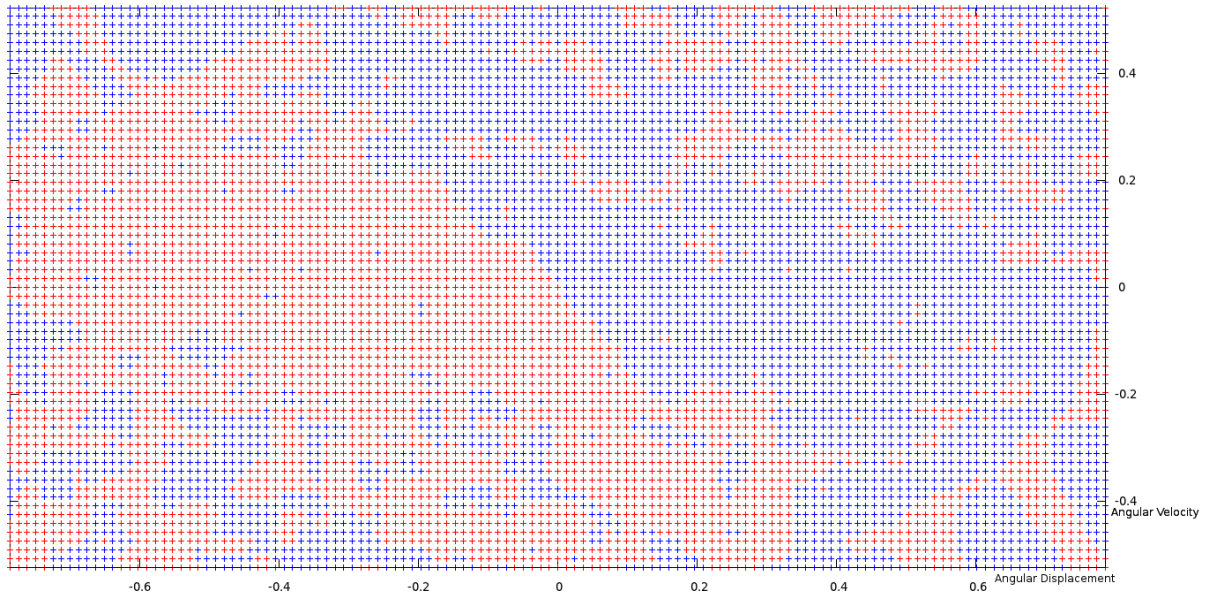


Figure 6.94: Another typical policy function for the X (roll) axis, after 5 million steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.



Figure 6.95: A typical Q-Function for the X (roll) axis, after 5 million steps (using  $\alpha = 0.3$ ,  $\gamma = 0.8$  and  $\lambda = 0.7$ ). (Angular) displacement in radians, velocity in radians per time-step.

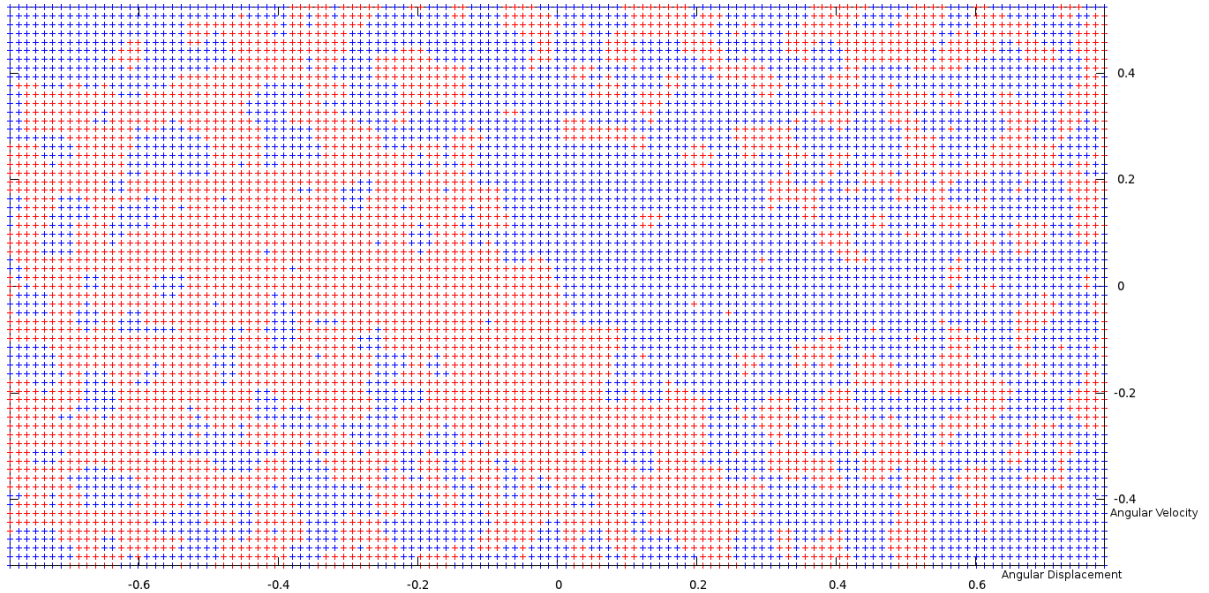


Figure 6.96: A typical policy function for the X (roll) axis, after 5 million steps (using  $\alpha = 0.3$ ,  $\gamma = 0.8$  and  $\lambda = 0.7$ ). (Angular) displacement in radians, velocity in radians per time-step.

Additional Q-function and policy plots are shown in Appendix A; a time series of plots taken during the course of a single trial is shown. This may be of interest to allow visualising how the Q-function and policy develop during the early phases of each trial.

## 6.6 Assessment of Results

The experiments detailed in §6.2 clearly showed that the Reinforcement Learning based controller is able to deliver stable motion control, without any underlying knowledge of the dynamics of the control plant. However, this performance has serious caveats: whilst the controller always *eventually* stabilizes the aircraft at the nominal position, the time taken to do this varies considerably from episode to episode, beyond the variation expected due to the random selection of initial states in each episode. Whilst in some cases, the time-domain response of the controller is almost indistinguishable from that of the PID controller used for comparison, in some episodes, the controller reacts badly, initially driving the aircraft *away* from the nominal state. This is understandable, given the nature of the Reinforcement-Learning algorithm involved.

Typically, the learning process has mostly stabilized within the first one or two million controller steps (11 – 22 hours of simulated operation), clearly visible in the trend of episode lengths tapering asymptotically towards a steady-state median value. There is still considerable variation in these episode lengths; in all the trials except those involving the presence of disturbances, the standard deviation of episode lengths for the Q-Learning based controller is more than that for a conventional PID controller. However, in most trials, the performance of the controller (particularly in terms of standard deviation of episode lengths) continues to improve up until the end of the simulation, which suggests that the learner has not yet fully learned an optimal policy. If the simulations were continued for longer, the performance would probably continue to improve (albeit at a progressively slower rate); this would most likely be visible in the form of a reduced frequency of unusually long episodes, as sensible behaviour is discovered even for states which are not visited frequently.

The experiments performed in §6.2 reveal that the effect of variations in controller parameters is complex to characterise; some parameters have significant effect on the learning or time-domain response performance of the controller, some have little effect, and some parameters have little effect over some stable range, but performance degrades rapidly outside these ranges. Some observations made regarding the selection of appropriate values are given in §8.1.

Certainly in some configurations, the Reinforcement-Learning based controller, once allowed time to stabilize, delivers a time-domain response which for some initial conditions, is as good as that of a reasonably well-tuned PID controller. However, even in these configurations, the RL based controller is generally unable to offer the same level of consistency as the PID controller, with some episodes having a very sub-optimal time-domain response. The exception to this is in the presence of considerable random disturbance; in this case, the RL based controller did not degrade as severely as that of the PID controller, which results in the RL controller delivering significantly better performance than the PID controller. This may be somewhat due to the limited resolution used by the Q-function approximator in this particular algorithm, and the use of bang-bang control.



## Chapter 7

# Computational Considerations

In this chapter, consideration is given to a different aspect of the viability of the RL based control system: the feasibility of implementing such a controller in an embedded computing environment. A series of tests to evaluate the computational viability of the controller are detailed, and the results of these tests examined.

### 7.1 Introduction

#### 7.1.1 Importance of Computational Performance

When evaluating the suitability of an algorithm for a specific task, pure evaluation of the ‘optimality’ of the outputs from the algorithm is insufficient; the difficulty of implementing the algorithm, in the context in which it will be *operationally deployed*, must be considered.

A number of modern control systems are implemented in an embedded context; accordingly the control algorithms used need to be designed keeping in mind the resources available in an embedded environment. This may impose restrictions regarding memory footprint, hardware features or computation time.

Many motion control systems operate on a fixed period, determined by the developer. Each time-step, the controller must evaluate its sensor inputs, process these inputs according to the control algorithm in use, then update the actuator outputs. Accordingly, hard real-time constraints are placed on the implementation of the control algorithm; failure to complete even a single iteration of the algorithm in the allocated time may result in control failure.

Even in the case where the motion control system operates on a variable period, there may still be some upper limit, beyond which the controller may fail. This still imposes hard real-time constraints on the operation of the control algorithm. It is thus clear that not only must the *mean* execution time of the algorithm be below a maximum threshold, but so must the *maximum* execution time for any one iteration.

This translates very clearly to the domain of this project: MAVs have very limited payload capacity, which imposes stringent restrictions on the processing capability of any prospective flight controller. If the machine learning algorithms utilized by this project are to be viable candidates for use in a flight controller, it must be clearly demonstrated that they are able to run on such devices, in real-time, at a frequency sufficiently high for smooth flight control.

This chapter offers evaluation of how well the machine learning algorithms used meet this requirement with contemporary embedded computing technology.

### 7.1.2 Practical Considerations

As mentioned, it is important to evaluate the viability of an algorithm destined for an embedded context in terms of the performance capabilities of the environment in which it will be deployed. However, for practical reasons, it may not be possible to perform the complete development cycle using the computing technologies to be used in the embedded environment.

For instance, due to the long run-times of the simulations performed as part of this project, experiments were accelerated by a fixed scale factor. Additionally, to further reduce run-times, the majority of experiments were performed on a desktop computer of a class substantially faster than expected in the final embedded system aboard a MAV.

To allow for comparison, and to assess the viability of using machine learning in practical control systems, an experiment was performed to characterise the run-time performance of the developed control system in both desktop environment and final embedded environment.

## 7.2 Computing Hardware Technologies

As stated in §7.1.2, two different computing hardware technologies were involved in the project. Experiments were performed on both a powerful desktop computer (to reduce simulation times) and an embedded computer (of the sort which could be readily integrated into a MAV).

### 7.2.1 Desktop Computer

The desktop computer used for the project is a Hewlett-Packard Z200 Workstation [3], a fairly typical commercial desktop workstation, in a mini-tower form-factor.

The Z200 is built around an Intel Core i5-660 CPU, which offers two identical cores using x86-64 architecture, operating at 3.33GHz. The CPU nominally offers Hyper-Threading, for a total of four logical cores, and ‘Turbo-Boost’ to a maximum of 3.6GHz. However, these two features were disabled in BIOS, so the CPU is considered a regular dual-core CPU operating at 3.33GHz. Each core has 64kB of L1 cache, 256kB of L2 cache, plus 4MB of L3 cache (or ‘Intel Smart Cache’) shared between the cores. The processor is connected to 4GB of dual-channel DDR3 SD-RAM at 1333MHz for main memory.

Storage is in the form of two 250GB 7200rpm SATA hard disk drives, however this is not of particular importance, since the software runs entirely from memory, without any significant swapping, and only interacts with permanent storage in order to write to the log file.

The computer runs Ubuntu Linux, Version 11.04 (‘Natty Narwhal’).

The Z200 comes with an nVidia Quadro NVS-295 graphics processor (GPU). However, the software used for this project does not utilize the graphics processor.

## 7.2.2 Embedded ARM Computer

The embedded computer used for the project is a PandaBoard, a community supported single-board computer intended as a development platform for OMAP4 based embedded systems [6]. Its primary use is the development of smart-phone technology, but its small form-factor coupled with powerful processor and convenient peripherals makes it suited for use in MAV research.

At the time of selection, the PandaBoard represented the most powerful off-the-shelf embedded processor suitable for use in a MAV environment. More powerful options are available in the form of x86-based single-board computers, but these tend to require heavy heat-sinks or fans, and have significantly greater power requirements, making them difficult to integrate into many MAV platforms.

The PandaBoard is built around a Texas Instruments OMAP4430 ‘application processor’ (which includes both CPU, GPU, common peripherals and memory in a single ‘package on package’ device). The OMAP4430 offers two identical cores using the ARM Cortex-A9 architecture, operating at 1.0GHz. Each core has 64kB of L1 cache, plus 1MB of L2 cache shared between the cores. There is also 48kB of on-chip RAM configured as L3 cache, presumed to be for the purpose of interaction with on-chip peripherals. The processor is connected to 1GB of dual-channel (low power) DDR2 SD-RAM at 400MHz for main memory.

Storage is provided in the form of a single 8GB class-ten SD-card, and additionally in the form of an 8GB USB FLASH drive.

The computer runs Ubuntu Linux, Version 12.04 LTS (‘Precise Pangolin’). This more recent version of Ubuntu is selected to utilize an updated version of the Texas Instrument peripheral drivers for the OMAP processor, thus improving stability.

Since the selection of the PandaBoard, a more powerful replacement, (the PandaBoard ES), has been released. The ES version increases the CPU clock rate from 1.0GHz to 1.2GHz, which would be expected to increase performance by approximately 20%. This highlights the rapid pace at which embedded processing technology is advancing.

Vital statistics for each of the two computing devices is shown in Table 7.1.

	CPU		Cache			RAM		
	Cores	Clock	L1	L2	L3	Clock	Channels	Capacity
<i>Units</i>	<i>Qty</i>	<i>GHz</i>	<i>kB</i>	<i>kB</i>	<i>kB</i>	<i>MHz</i>	<i>Qty</i>	<i>GB</i>
<b>Platforms</b>								
HP Z200	2	3.33	64	256	4096	1333	2	4
PandaBoard	2	1.00	64	1024	48	400	2	1
Difference Factor	1.00	3.33	1.00	0.25	85.33	3.33	1.00	4.00

Table 7.1: Summary of Vital Statistics for Computing Hardware Technologies.

## 7.3 Performance Comparison

### 7.3.1 Performance Metrics

The important question which needs to be answered in terms of determining the validity of running a particular algorithm on a particular hardware platform is ‘can the hardware run the implementation of the algorithm sufficiently quickly that time taken for each iteration of the algorithm is less than the maximum allowable time between controller updates?’ This can be expressed as in Equation 7.1.

$$t_{iterate} < t_{control} \qquad t_{iterate} < \frac{1}{f_{control}} \quad (7.1)$$

Where  $t_{iterate}$  is the *maximum* time per iteration,  $t_{control}$  is the *maximum* allowable time between controller updates, and  $f_{control}$  is the *minimum* allowable frequency of controller updates.

This is straightforward. Generally, if the software simply runs the algorithm as quickly as it can, then  $t_{iterate} = t_{algorithm}$ . Hence, the suitability of a hardware platform for a particular algorithm can be determined by measuring  $t_{iterate}$  and comparing it against the selected minimum value for  $f_{control}$ .

However, the simulator used for this project is designed to run at a set update frequency: in simulator-time, there will always be exactly  $f_{control}$  iterations per second, so  $t_{iterate} = \frac{1}{f_{control}}$ . In real-time, time between simulation iterations is given by Equation 7.2.

$$t_{iterate} = \begin{cases} t_{control}, & (t_{algorithm} + t_{overhead}) < t_{control} \\ (t_{algorithm} + t_{overhead}), & (t_{algorithm} + t_{overhead}) > t_{control} \end{cases} \quad (7.2)$$

Where  $t_{iterate}$  is the real-time between simulation iterations,  $t_{algorithm}$  is the real-time taken to process each iteration of the algorithm,  $t_{control}$  is the *fixed* simulator period for each iteration, and  $t_{overhead}$  is some small time required to update other elements of the simulation. This is further complicated by allowing the simulator to virtually compress time; to reduce the time taken for long simulations. In this case,  $t_{control}$  is first divided by a compression factor. Regardless, the important metric is  $t_{iterate}$ .

It is assumed that  $t_{overhead}$  is sufficiently small as to be negligible, so that if  $t_{iterate}$  is greater than  $t_{control}$ , then  $t_{iterate}$  may be used to provide a good estimation of  $t_{algorithm}$ . In this case, the simulator is configured to perform twenty-five ‘frames’ per second, and a compression factor of twenty is used; so  $t_{control} = 2ms$ .

The dynamics of a quadrotor MAV are reasonably slow (compared to the requirements imposed on many other modern digital control systems). Accordingly, the minimum acceptable frequency of controller updates will be fairly low. For this project, a value of  $20Hz$  was considered the minimum acceptable level. This was selected as being about five times slower than the rate at which common autopilots receive updated state information from their inertial measurement units [9], to allow for noise filtering. Thus, the maximum acceptable value of  $t_{algorithm}$  will be  $50ms$ .



### 7.3.2 Experimental Procedure

For each of the two hardware platforms, two experiments were performed: one experiment to characterise the performance of the hardware whilst running the machine learning based control algorithm (as per §4.3.2), and a second experiment to characterise the performance of the hardware whilst running a conventional PID control algorithm (as per §4.3.1), to use as a control. For each experiment, three trials were performed.

In each trial, the control algorithm was simulated for a duration of one million steps. This is shorter than the experiments performed to assess the control performance of the algorithms, since it was assumed that there would be no significant change in the computational load after the first million steps. The elapsed real-time between the completion of every successive thousand steps was logged by the simulator.

Following the completion of each trial, the log files were parsed to extract the relevant timing information, which was then analysed to assess the computational performance of the trial.

#### 7.3.2.1 Multitasking During Experiments

As both platforms are running operating systems with pre-emptive multitasking, and running a graphical desktop environment, there are a number of other processes (in addition to the Java process running the experiment) running on each machine during experiments. The experimental processes were run with a default ‘niceness’ (priority) of 0, giving them no higher priority than most other processes in a desktop environment. This could significantly affect the results of an experiment, if some other process were to consume a significant amount of processing resources during a trial.

However, after consideration, it was decided that any effects on the experimental outcomes would be negligible. The experimental software is all single threaded, and both computers use dual-core processors. During an experiment, the experimental process would generally consume one hundred percent CPU time on a single core, whilst the other core would remain largely unloaded. Based on the assumption that the experimental software was CPU bound, and that there was still adequate processing power remaining on the unloaded core to satisfy any other processes arising from the graphical desktop environment, the performance of the experimental software should be negligibly affected.

### 7.3.3 Expectations

Brief evaluation of Table 7.1 makes it clear that the desktop Z200 would be expected to offer higher performance than the PandaBoard. It is difficult to estimate exactly how much this difference will be, based on the published details of each device. Consider Table 7.1: the final row indicates the proportional difference between the two hardware platforms, in terms of each of the specified characteristics. If the performance of the software on each platform is directly proportional to each of the characteristics, and each characteristic is weighted evenly, the expected difference in performance between the two platforms would be given by Equation 7.3.

$$d = 1.00 \times 3.33 \times 1.00 \times 0.25 \times 85.33 \times 3.33 \times 1.00 \times 4.00 = 946 \quad (7.3)$$

This would suggest that the desktop Z200 machine would perform approximately 950 times faster than the embedded PandaBoard. Since it is extremely unlikely that the performance is an equally weighted function of each of the listed characteristics, this estimate has significant limitations, and is better treated as an upper bound on the difference. If instead, it is assumed performance is solely a function of CPU clock speed, then the desktop computer would be expected to be 3.33 times faster than the embedded device. Both of these estimates fail to take into consideration the efficiency of the CPU instruction set, which will be critically important in determining performance (in addition to any number of other factors).

Regardless, this provides an estimation of the expected bounds on the difference in performance to expect between the two platforms. It is almost certain that the performance on the embedded device will be *significantly* worse than that on the desktop workstation, and a difference by a factor of tens to hundreds would be reasonable.

## 7.3.4 Results

### 7.3.4.1 Desktop Computer

The total time take by the desktop computer to complete each experiment was averaged across the three trials performed, and the results are summarised in Table 7.2.

The desktop PC takes 2053 seconds (approximately 35 minutes) to perform one million steps using the basic PID controller; this corresponds to an average of around  $2ms$  per simulation step, which indicates that  $t_{algorithm} < t_{control}$ , so  $t_{algorithm} < 2ms$ . When using the Q-Learning algorithm, execution time is increased to 9942 seconds (two hours and forty five minutes); this corresponds to an average of approximately  $10ms$  per simulation step, which indicates that  $t_{algorithm} \approx 10ms$ .

In both these cases,  $t_{algorithm}$  is under the maximum acceptable value of  $50ms$ , which suggests that the desktop computer has sufficient performance to run both these algorithms in a real time control environment.

Because the PID algorithm does not exhibit any learning behaviour, it is expected that execution time remains constant throughout each experiment. Conversely, the computational requirements of the Q-Learning algorithm increase as the algorithm learns the environment, due to the increasing size of the adaptive RBF network. These characteristics are visible in Figure 7.1.

### 7.3.4.2 Embedded ARM Computer

The total time taken by the embedded ARM computer to complete each experiment was averaged across the three trials performed, and the results are summarised in Table 7.2.

Hardware	Controller	Total Time (s)
Desktop	PID	2053
Desktop	Q-Learning	9942
Embedded	PID	7923
Embedded	Q-Learning	689647

Table 7.2: Total execution time for one million steps of simulation.

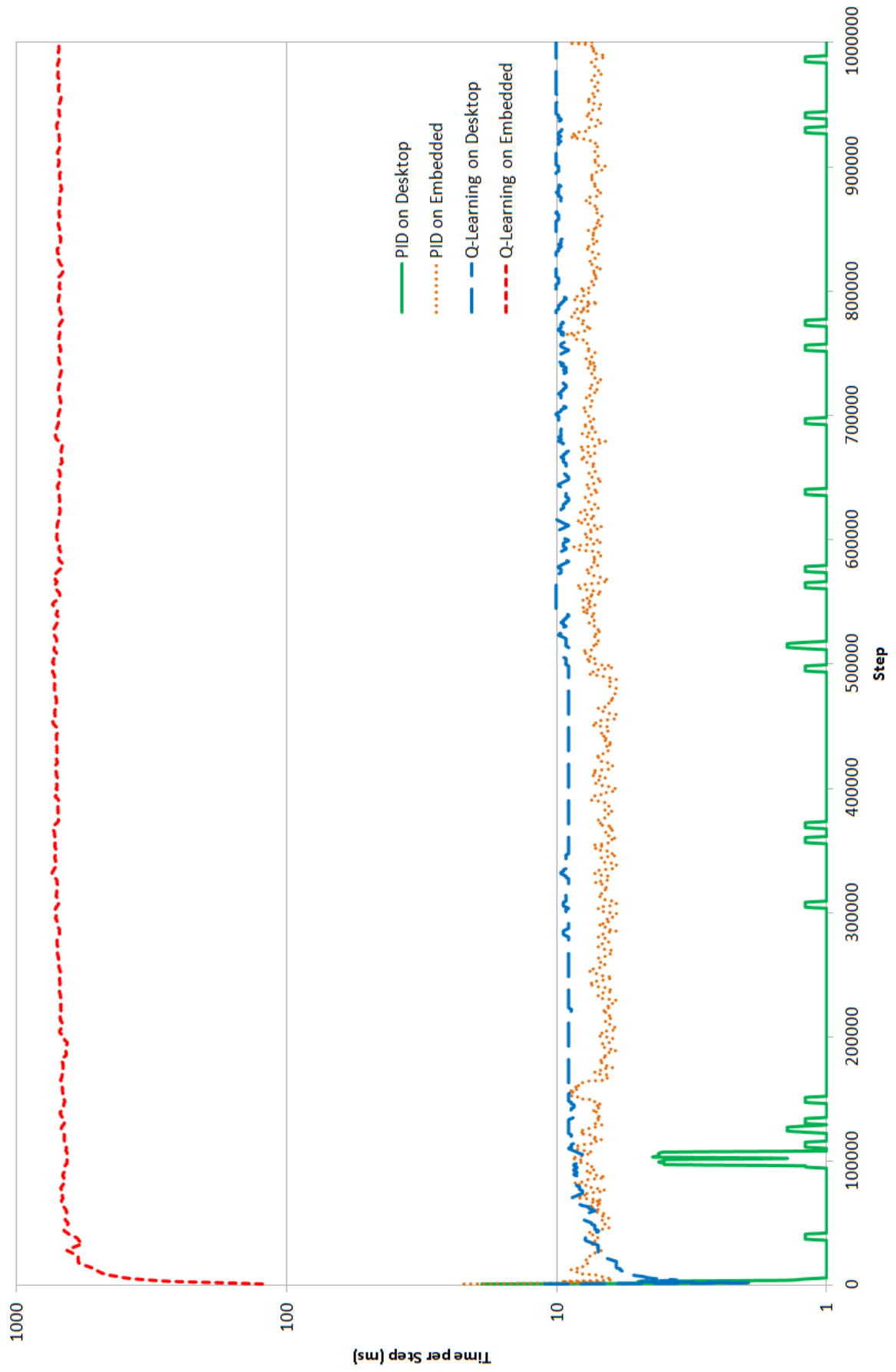


Figure 7.1: Time between Simulation Iterations

The embedded computer takes 7923 seconds (approximately two hours and fifteen minutes) to perform one million steps using the basic PID controller; this corresponds to an average of around  $8ms$  per simulation step, which indicates that  $t_{algorithm} \approx 8ms$ . This suggests that the desktop PC is *at least* four times faster than the embedded ARM computer. When using the Q-Learning algorithm, execution time is increased to 689647 seconds (approximately eight days); this corresponds to an average of around  $690ms$  per simulation step, which indicates that  $t_{algorithm} \approx 690ms$ . This suggests that the desktop PC is 69 times faster than the embedded ARM computer.

Whilst running the PID algorithm,  $t_{algorithm}$  is under the maximum acceptable value of  $50ms$ , which suggests that the embedded ARM computer would have sufficient performance to run this algorithm in a real-time control environment. However, whilst running the Q-Learning algorithm,  $t_{algorithm}$  is significantly greater than the maximum acceptable value, which suggests that the embedded ARM computer would have insufficient performance to be able to run the algorithm reliably in a real-time control environment.

Because the PID algorithm does not exhibit any learning behaviour, it is expected that execution time remains constant throughout each experiment. Conversely, the computational requirements of the Q-Learning algorithm increase as the algorithm learns the environment, due to the size of the adaptive RBF network increasing. These characteristics are visible in Figure 7.1. Specifically, the time per iteration whilst running the Q-Learning algorithm can be clearly seen to increase during the early stages of the experiment, before reaching a steady state of around  $700ms$  per iteration.

## Chapter 8

# Conclusions

In this chapter, the overall outcomes from the project are summarised. Consideration is given to possibilities for extending this line of research in the future. Additionally, a set of guidelines are proposed, which can be of utility of developers looking for information on suitable parameter values for similar RL control systems.

### 8.1 Considerations for Parameter Selection

Based upon the results from §6.2, the following observations were made regarding the selecting of parameter values for a Q-Learning based control system.

Note that these conclusions are based on the results from this project, and may not translate into rules for selecting controller parameters which deliver good performance for any other control application.

**Selecting  $\alpha$  (learning rate)** As identified in §6.2.2, there is a clear compromise between the rate at which learning occurs, and the stability of the resulting controller behaviour. Hence, to select a value for  $\alpha$ , the maximum time allowable before the controller stabilized into a reasonably steady state (i.e. a maximum length for regions one or two) should be chosen. Then, the minimum value of  $\alpha$  which allows the controller to meet this criteria should be used. This delivers the most stable behaviour over the long term, whilst learning adequate behaviour at sufficient speed.

**Selecting  $\gamma$  (discount rate)** As identified in §6.2.3, selection of  $\gamma$  value is not critically important, so long as the value is within a specific range: performance in terms of both learning rate and median episode length, improve with increasing  $\gamma$ . However, the improvement is less pronounced for higher values, and further, at high  $\gamma$  values, the controller stability decreases substantially (evidenced by increasing standard deviation of results). This results in a ‘plateau’ in which small changes in  $\gamma$  have little affect on overall performance; best practice would be to select a value for  $\gamma$  which is known to fall into this region (in this case between around 0.85-0.95), since there are only marginal gains to be had in more specific selection.

**Selecting  $\lambda$  (eligibility trace decay rate)** A similar situation applies for the selection of  $\lambda$ . As identified in §6.2.4, controller performance improves with increasing  $\lambda$ . However, at high values for  $\lambda$  there is a decrease in stability early in the learning process. This decrease appears less abrupt than that due to increasing the parameter  $\gamma$  to too high a value. Again, best practice would be to simply select a value for  $\lambda$  which is known to be in the stable region (in this case between around 0.8-0.95), since there are only marginal gains to be had in more specific solution.

**Selecting  $K_{\dot{\theta}}$  (velocity reward gain)** As identified in §6.2.5, selecting an appropriate value for velocity reward gain is critical to obtain optimal performance from the control system. There is little in the way of trade-off; the controller performance is simply better for the correct value of  $K_{\dot{\theta}}$ . In this case, there is a maxima located somewhere between 0.2 and 0.35. Best practice would be to conduct a number of trials to determine the closest match to the optimum value for  $K_{\dot{\theta}}$ .

**Selecting  $\sigma_{\theta}$  and  $\sigma_{\dot{\theta}}$  (ARBFN resolution)** As identified in §6.2.6 and §6.2.7, there is substantial degradation in performance if the ARBFN resolution selected is too low; the low resolution experiments were the only experiments in the course of this project in which the learner failed to settle to a policy which behaved similarly to that of a conventional control system. However, there is little benefit from an unnecessarily high resolution, and higher resolutions incur substantial increase in computational requirements. Hence, best practice would be to select the lowest resolution which still offers acceptable behaviour.

**Selecting  $\epsilon$  (greediness)** As identified in §6.2.8, there is a clear compromise between the rate at which learning occurs, and the stability of the resulting controller behaviour. This is similar to the performance affect of varying the learning rate  $\alpha$ . A similar mechanism for choosing a value for  $\epsilon$  should be utilized: select the smallest value of  $\epsilon$  which results in the controller settling into a steady state in an acceptable length of time.

## 8.2 Viability for Practical Use

The testing performed in this project clearly demonstrates that, under some circumstances, a Q-Learning based motion control system can offer performance as good as a conventional PID control system. However, in order for the Q-Learning based controller tested in this project to be considered a possible alternative to a conventional PID controller, it would need to demonstrate some clear benefits over a PID controller. Such benefits were proposed in §1.1.3: machine-learning could alleviate the need for advance knowledge of the system dynamics, and ML could compensate for changes in dynamics at run-time.

The testing of performance for different variations in controller parameters performed in §6.2 reveals that changes in parameters to the control system may result in substantial changes in performance. This is not desirable, since the need to tune parameter values made conventional PID controller design inconvenient, so would not represent an advantage to the Reinforcement Learning controller design. Arguably, many of the parameters only characterise the learning behaviour of the controller, and given sufficient time, controllers with different learning behaviour will eventually reach (approximately) the same steady-state policy. Hence, the only criteria is

the selection of stable parameters, so that the controller eventually reaches the desired steady-state. However, the values used in the reward function for the learner have a considerable affect on the steady-state performance of the controller, and these must be tuned specifically to obtain the desired performance. The sensitivity of the controller to these parameters makes it not substantially more convenient than a conventional PID controller.

The testing of performance in the face of changes in dynamics at runtime indicates that the Q-Learning controller may have advantage over the conventional PID controller in this area. The median performance of the Q-Learning controller and PID controller are approximately equivalent, both before and after the dynamics change. However, the standard deviation of episode length, whilst substantially in favour of the PID controller before the change in dynamics, is approximately even after the change. Since the change in dynamics occurring in these experiments is small, it is possible that in the event of a severe change in plant dynamics, the Q-Learning controller would display a distinct advantage.

Of course, the Q-Learning system has an obvious disadvantage, compared to a conventional model-based control system: the learning control system takes some time to learn the dynamics of the environment. During this time (which may be many hours in a real world controller), the controller behaviour is poor; often driving the output until it encounters some boundary condition. Examples of poor behaviour can persist for a long time, if the controller finds itself in states which are uncommon, and have not yet been ‘explored’. This alone may make this type of controller unsuitable for a range of motion control tasks, in which it is not feasible to allow the controller sufficient time to learn. Many boundary conditions result in damage to equipment and must be strictly avoided. Clearly, if this sort of controller were simply implemented directly on an unrestrained quadrotor MAV, the vehicle would most likely be severely damaged in the first few minutes of operation.

Finally, the Q-Learning methodology has a further disadvantage in terms of computational requirements. As detailed in §7, running in an embedded environment, the Q-Learning control algorithm takes around 690ms per iteration, against 8ms for a conventional PID calculation. This severely affects the practicality of using the Q-Learning algorithm in an embedded control environment: even on a processor which represented the highest available performance in an embedded device at the time of the experiment, the algorithm would be unable to keep up with a control system updating at 20Hz.

All in all, the Q-Learning algorithm offers interesting performance characteristics, some of which could be desirable in specific control applications. However, practical considerations; computational load, long learning times, and sensitivity to parameter selection; mean that the concept does not represent a compelling advantage over more conventional controller designs at this time. It is possible that in the future, improved Reinforcement-Learning algorithms, coupled with higher performance embedded computing hardware, will result in a refinement of this concept becoming a viable motion controller design.

### 8.3 Further Expansion & Future Work

Whilst this project has delivered important insights into the viability of using a Reinforcement-Learning based control algorithm in an embedded motion control application, there are aspects into which continued research would enhance understanding of the concept:

### 8.3.1 Longer Simulations

In most of the experiments, the performance metrics of the Q-Learning based controller continue to show discernible learning behaviour until the end of the experiment: performance in region four of many experiments is better than that of region three. This suggests that the performance of the controller would continue to improve if the simulations were allowed to run longer. This was not feasible within the time-frame and scope of this project.

It would be desirable to conduct a series of similar experiments, with a longer simulation run-time; twenty million steps (each simulation being four times longer) might be a suitable starting point. This would provide evidence of what the maximum level of performance that can be attained with this type of controller.

### 8.3.2 More Comprehensive Test Regime

Prolonged simulation run-times created constrained the number of trial opportunities, meaning the experiments performed as part of this project lack statistical rigour. Both the environment and controller display stochastic behaviour, and obtaining an accurate understanding of the expected behaviour of the algorithm depends upon the law of large numbers. Larger sample sizes than those in this project may offer a higher degree of confidence that the observed behaviour of the algorithm is entirely representative of the expected behaviour.

Any limitation in the study is compounded by the intention of this project to deliver a broad overview of the practical implications of this type of control algorithm. Thus, experiments were unable to concentrate on as small a subset of the possible algorithmic configurations as might be necessary in a study whose purpose was to deliver a high resolution depiction of performance as a function of configuration. Even then, this project only utilized a single Reinforcement-Learning algorithm; one of many.

To address this limitation, it would be desirable to conduct a more comprehensive study of this type of algorithm: one which delivered more reliable results through an increase in sample size (the number of repeated trials for each experimental configuration) and scope. An increase in sample size would improve confidence in the expected behaviour of an algorithm, whilst an increase in scope would allow higher resolution information on the effects of specific configuration variations on performance, and consideration of other kinds of reinforcement learning algorithms.

### 8.3.3 Testing on Test Platform

This project was a simulation based exercise. This was necessary given the time available, but the simulation represents only a simple approximation of the dynamics of a ‘physical’ aircraft. In order to fully qualify the concept of a Reinforcement-Learning based motion controller as viable for practical use, it would be necessary to test the controller on physical plant.

With this in mind, the software created for use in this project was designed to allow easy transition from simulated operation to control of a real aircraft. As detailed in §3.1.4, a UAV Test Stand is available in the Mechatronics Laboratory at the University of Canterbury, and can be configured to behave in a manner similar to the simulated UAV plant used for this project.



It would be desirable to perform testing of the Q-Learning based control system on a quadrotor UAV mounted on the UAV test stand; this testing and subsequent analysis should follow a procedure similar to that outlined in §6. Comparison between these results, obtained using real physical plant and those obtained in this project using simulation, would allow assessment of the viability of the controller concept, as well as validating the results from this project.

#### 8.3.4 Free Rotation

As detailed in §3.1.4, design considerations in the UAV Test Stand prevent the vehicle from undergoing unrestricted rotational motion. In order to facilitate future testing of the control system on the UAV Test Stand (as per the previous section), the simulator was designed to reflect these restrictions (see §4.2.2).

Such restrictions would not be present on any free-flying MAV. In order to accurately assess how the controller would perform on a real aircraft, it would be necessary to test the controller in a regime which allows unrestricted 360° rotation about all three axes. This could be performed first in simulation, by reconfiguring the simulator software. Further testing on physical plant would depend upon continued development of the UAV Test Stand to allow continuous rotation: the current development road-map for the Test Stand includes the addition of continuous rotation functionality about the yaw axis, though extending this to the roll and pitch axes would require further consideration.

#### 8.3.5 Incorporating Translational Control

As detailed in §1.2.3, the control task considered in this project was a considerable simplification of the problem faced when designing an actual flight control system for a multirotor MAV: a MAV operates with six (highly coupled) degrees of freedom. This project considered only the three rotational degrees. Additionally, the three rotational degrees of freedom which were considered, were treated as completely decoupled, thus differing from the true nature of a multirotor MAV.

In order to factually assess the viability of this kind of controller for use in multirotor flight control applications, the controller would be tested in an environment which matches the true dynamics of a multirotor MAV as closely as possible. This would require reconfiguration of the simulator to allow translational motion, and reconfiguration of the controller to utilize a single learner instance to operate on all six axes.

The extension of the learner component to operate in higher dimensions will have a considerable affect on computational performance. Since (as evidenced in §7) the configuration of algorithm used in this study is already too computationally intensive for common embedded controllers, a substantial improvement in performance will be necessary before testing in higher dimensional environments becomes relevant in practical terms.

#### 8.3.6 Adding Multiple Actions & Continuous Actions

This project restricted its scope to two available actions per state: in each axis, full control authority in one direction or the other (bang-bang control). This was a concession to reduce learning times, since with more possible actions, more trials are required to determine the

optimum output for a given state. In reality, most motion control systems utilize either multiple levels of controller output, or continuously variable controller output. Improved resolution of controller output should deliver improved controller performance (for small error values when the output is not saturated).

Increasing the number of available actions per state is a straight-forward method of increasing the resolution of controller output, but at the cost of learning time. However, obtaining fully continuous controller output is complicated, since the Q-Learning algorithm used here is designed for operation with discrete state/action pairs. One possible way to address this may be through the use of fuzzy logic techniques.

### 8.3.7 Hybrid Approaches

The results obtained during this study, reveal that the primary disadvantage of the Reinforcement-Learning based controller is the poor controller performance which occurs initially. Because the controller starts with no information about the environment, it is unable to control the aircraft, until it has gathered sufficient information on the dynamics of the aircraft. Further, the performance of the controller may, for some choice of initial conditions, remain poor for a long time. Whilst the environment behaviour for states which are close to the nominal state is well observed, states which are far from the nominal state are not visited as frequently. Thus obtaining sufficient detail to form a policy which behaves well in these states takes longer.

One possible mechanism for addressing these deficiencies is through the use of ‘Learning by Demonstration’. In Learning by Demonstration, the learning controller observes the behaviour of some external controller for a number of trials. The Learning by Demonstration controller uses an algorithm to generalize the observed behaviour. This generalized behaviour is then followed by the controller, to replicate the behaviour of the external controller.

A pure Learning by Demonstration based controller would not encapsulate the desired benefits of a Reinforcement Learning based controller, since this type of controller only learns from the exemplar behaviour it observes, and not from its own interactions with the environment. However, it may provide tangible benefits if used as part of a hybrid design:

In such a hybrid controller, the Learning by Demonstration would initially be used to generalise the behaviour observed from a set of exemplars, as performed by some conventional controller (which might offer stable, but otherwise suboptimal, performance). Once the learning period is complete, the generalised policy would be the basis for the reinforcement learning component of the controller. This would allow the controller to improve upon the suboptimal behaviour initially observed, whilst avoiding the initial wild flailing which makes a pure Reinforcement Learning approach difficult to consider seriously.

In the simplest case, this sort of hybrid behaviour could be obtained fairly easily by configuring the software so the controller alternates between using a policy informed by the Q-Learning agent, and that of a conventional PID controller. The learner component of the agent would observe the behaviour resulting from both policies, thus addressing the issue of taking a long time before ‘stumbling’ upon stabilizing behaviour for states being seldomly visited. The class `PolicyMux` was created to facilitate future experiments of this type.

## 8.4 Closing Remarks

The creation of a complex motion control system, such as that required as a flight controller in an unmanned aerial system, presents a number of challenges. The need for a good model of the dynamics of an aircraft, in order to develop a control system which implements appropriate response behaviour, can substantially increase the development time required, especially if non-trivial changes to the aircraft (which may affect its dynamics) are made throughout the development process.

Reinforcement Learning offers a variety of compelling possibilities. The ability to learn the dynamics of an environment ‘on the fly’ suggests that a Reinforcement Learning based algorithm may offer a mechanism to address some of the difficulties commonly encountered by those faced with developing control systems.

The intention at the outset of this project has been successful. It has clearly demonstrated that a Reinforcement Learning based motion control system can offer control performance on par with a conventional PID controller design, without the controller having prior knowledge of the system dynamics. The study shows explicitly that this advantage comes at significant cost; whilst *typical* performance may be excellent, such a controller struggles, on the time-scales tested here, to provide reliable behaviour across its entire workspace. Thus confirming a major liability for motion control applications, especially unmanned aerial vehicles.

The project not intended to be comprehensive. Its aim was to provide a highly accessible guide to the viability of Reinforcement Learning in particular applications. A number of specific areas for further investigation were identified, which may offer a fuller picture of the performance to be expected from a Q-Learning algorithm (or variants thereof). The results captured are expected to enable an informed decision about the practicality of using these sort of algorithms in a control system.

Whilst a Reinforcement Learning based control system, as outlined in this project, offers benefits with regards to common issues in control system design, study has revealed a number of disadvantages. This thesis contends that a more protracted inquiry is needed before a complete understanding of the viability of the concept is developed. Further, it is contended that further improvements in embedded computing performance and machine learning algorithm design are required in order to warrant considering implementation of such a system in a practical application.



## Appendix A

# Additional Q-Function and Policy Plots

Shown on the following pages are Q-function and policy plots, collected sequentially during the course of a single trial. These may be used to better visualise how the Q-function (and hence policy) develops during the learning process. Plots are shown after 10, 20, 30 and 50 thousand steps; then after 100, 200, 300, 500 thousand steps; and finally after 1 million steps.

As expected (see §6.5 for more details), the Q-function begins uniformly, before developing a well defined ‘ridge’ around the nominal position. The policy begins with the entire state-space being ‘patchy’, before developing an identifiable separation along the ‘ridge-line’ after around 200,000 steps.



Figure A.1: A typical Q-Function for the X (roll) axis, after 10,000 steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). X-axis in Radians, Y-axis in Radians per time-step.

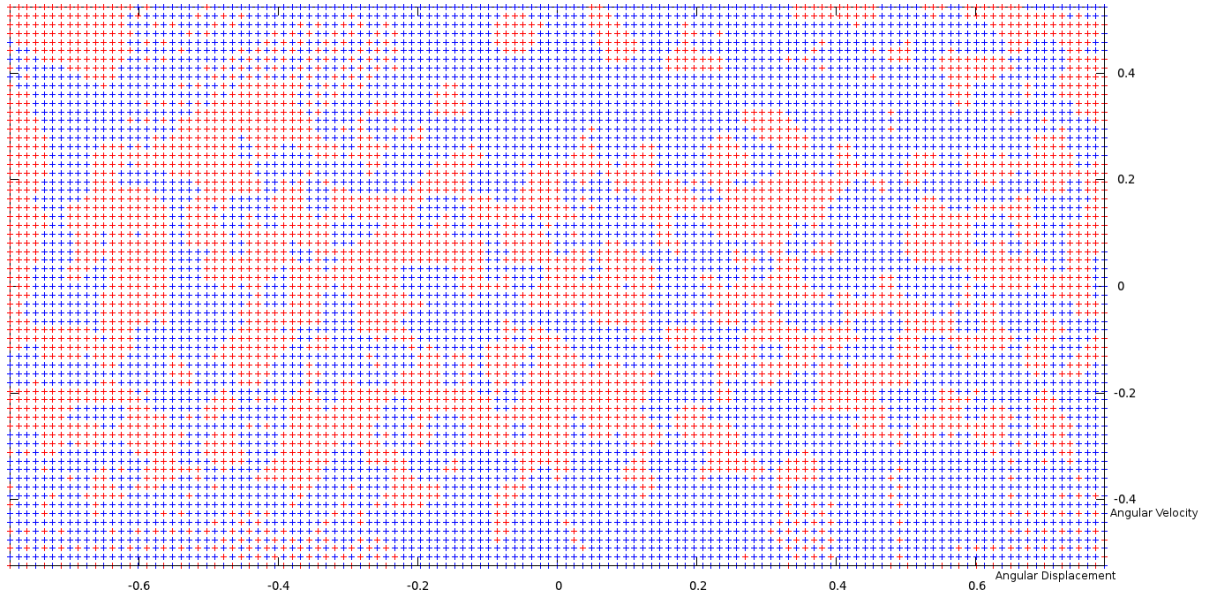


Figure A.2: A typical policy function for the X (roll) axis, after 10,000 steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.



Figure A.3: A typical Q-Function for the X (roll) axis, after 20,000 steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.

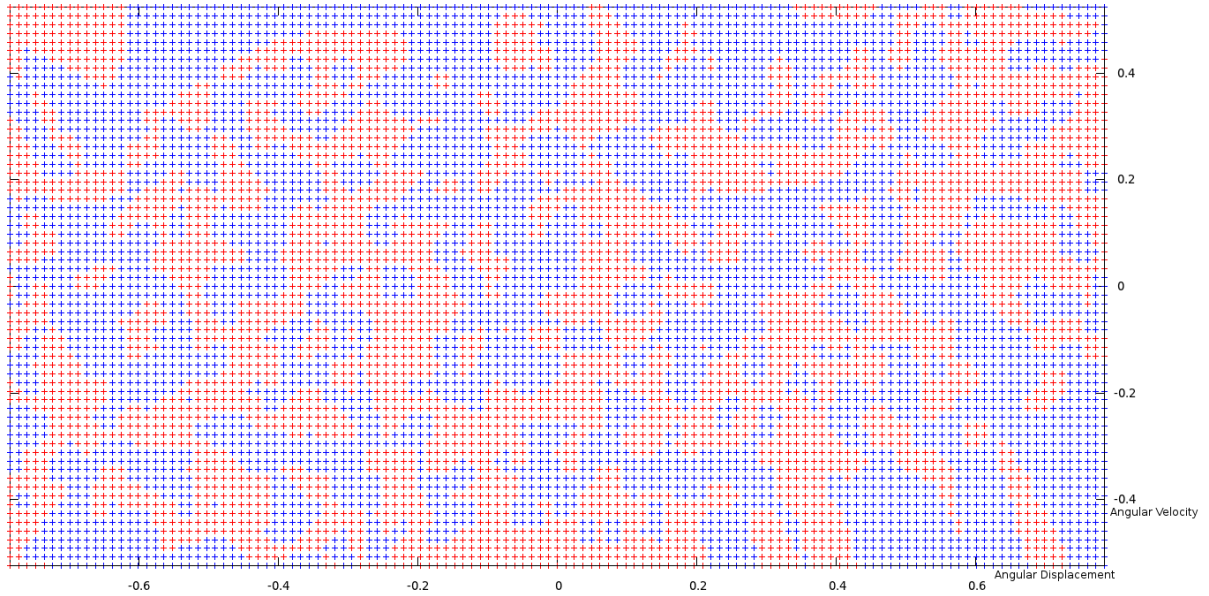


Figure A.4: A typical policy function for the X (roll) axis, after 20,000 steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.



Figure A.5: A typical Q-Function for the X (roll) axis, after 30,000 steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.

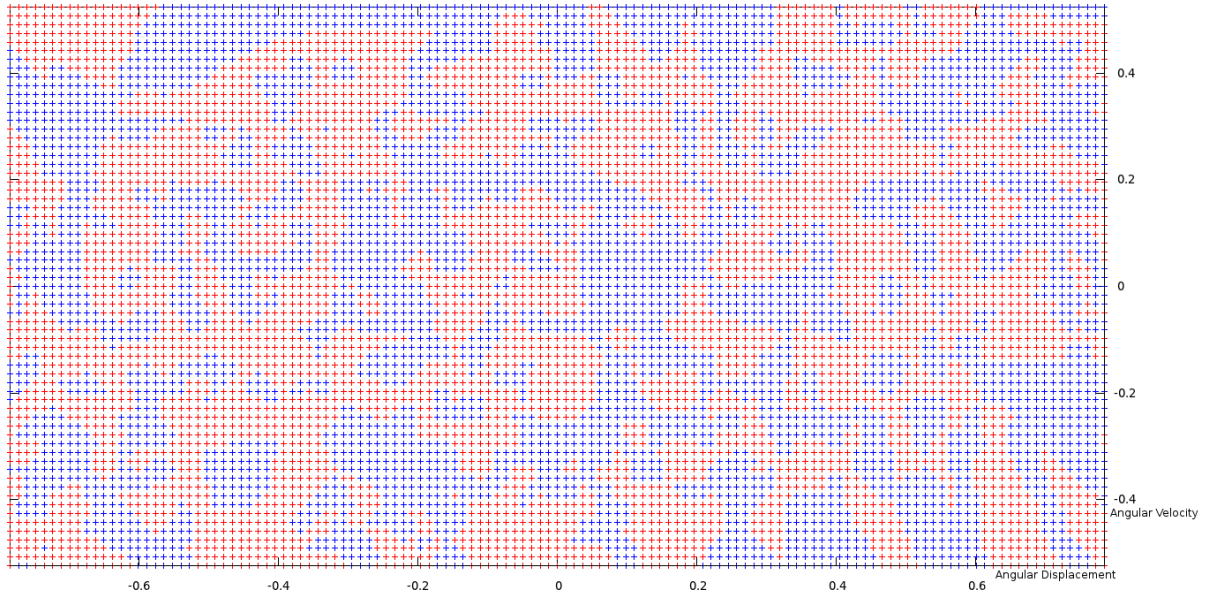


Figure A.6: A typical policy function for the X (roll) axis, after 30,000 steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.





Figure A.7: A typical Q-Function for the X (roll) axis, after 50,000 steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.

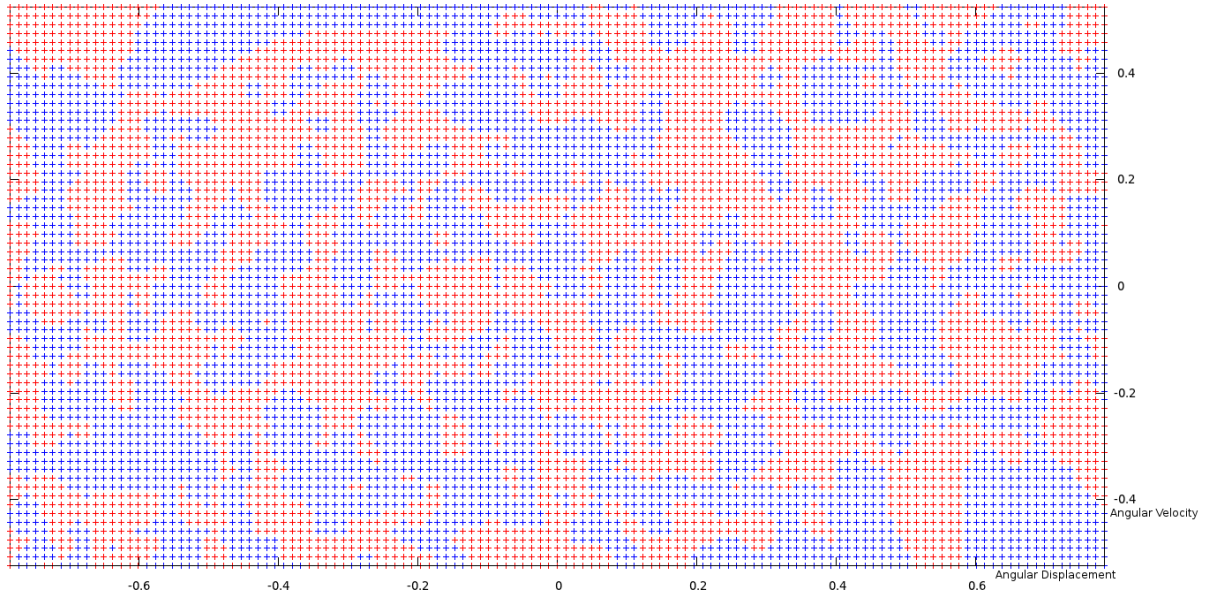


Figure A.8: A typical policy function for the X (roll) axis, after 50,000 steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.



Figure A.9: A typical Q-Function for the X (roll) axis, after 100,000 steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.

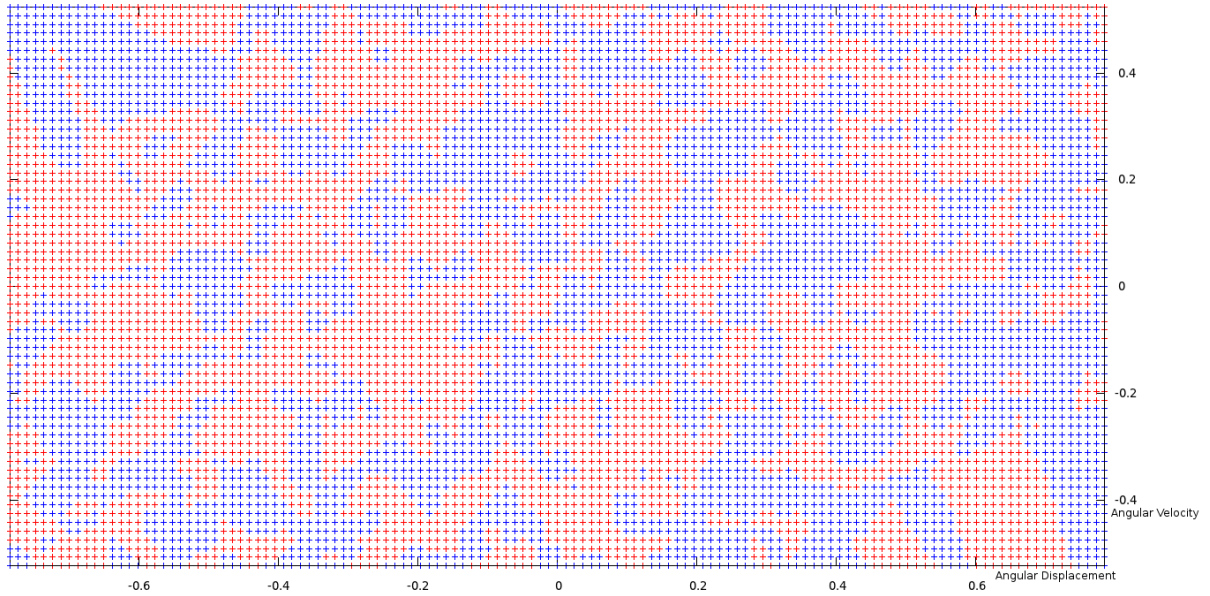


Figure A.10: A typical policy function for the X (roll) axis, after 100,000 steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.



Figure A.11: A typical Q-Function for the X (roll) axis, after 200,000 steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.

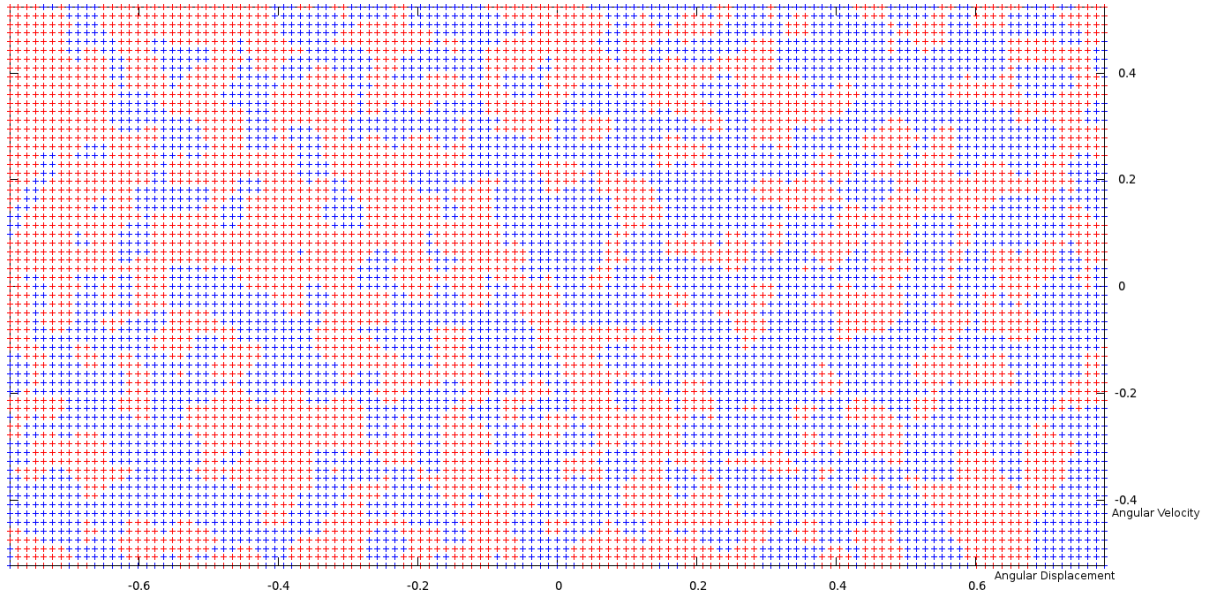


Figure A.12: A typical policy function for the X (roll) axis, after 200,000 steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.



Figure A.13: A typical Q-Function for the X (roll) axis, after 300,000 steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.

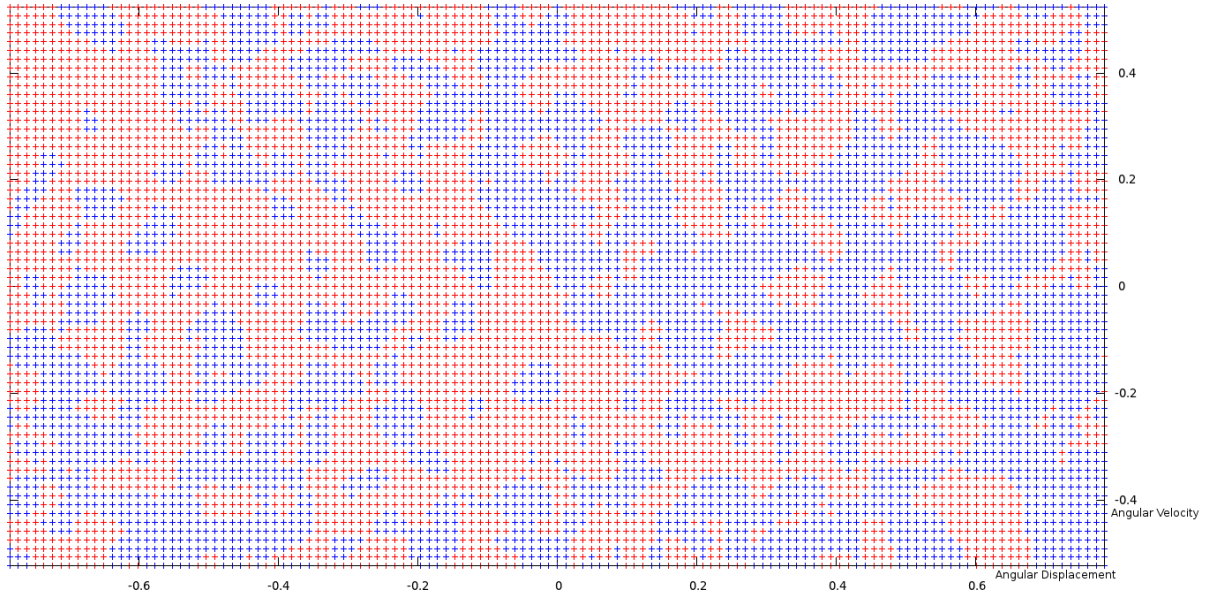


Figure A.14: A typical policy function for the X (roll) axis, after 300,000 steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.





Figure A.15: A typical Q-Function for the X (roll) axis, after 500,000 steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.

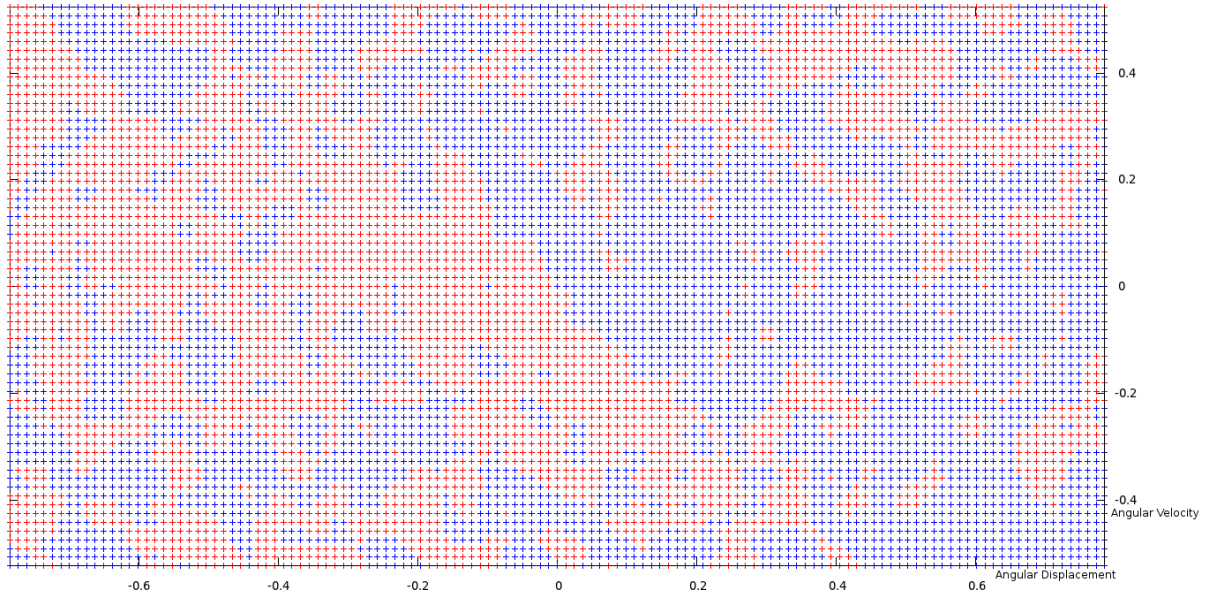


Figure A.16: A typical policy function for the X (roll) axis, after 500,000 steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.



Figure A.17: A typical Q-Function for the X (roll) axis, after 1000,000 steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.

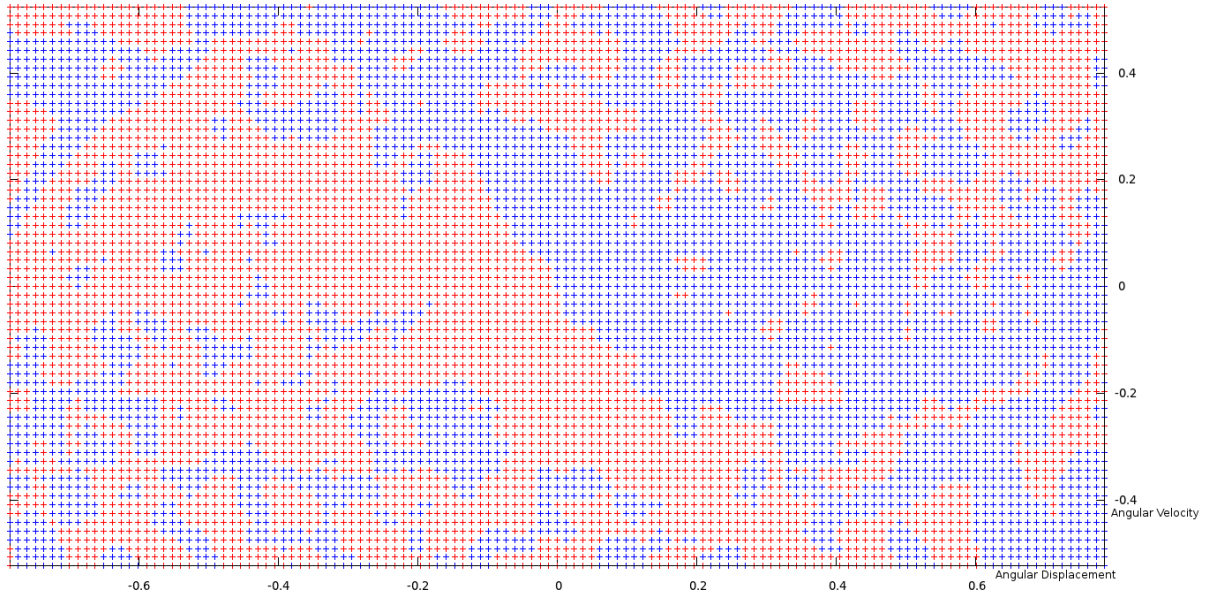


Figure A.18: A typical policy function for the X (roll) axis, after 1000,000 steps (using  $\alpha = 0.2$ ,  $\gamma = 0.9$  and  $\lambda = 0.8$ ). (Angular) displacement in radians, velocity in radians per time-step.

# Bibliography

- [1] MASON project page. <http://cs.gmu.edu/~eclab/projects/mason/>.
- [2] International journal of micro air vehicles. <http://multi-science.metapress.com/content/121501/?sortorder=asc>, 2009.
- [3] HP Z200 workstation datasheet. [http://h10010.www1.hp.com/wwpc/pscmisc/vac/us/product\\_pdfs/Z200\\_datasheet\\_011309\\_highres.pdf](http://h10010.www1.hp.com/wwpc/pscmisc/vac/us/product_pdfs/Z200_datasheet_011309_highres.pdf), March 2010.
- [4] The player project. <http://playerstage.sourceforge.net/>, 2010.
- [5] RL-Glue: Main page. [http://glue.rl-community.org/wiki/Main\\_Page](http://glue.rl-community.org/wiki/Main_Page), 2010.
- [6] Pandaboard board references. <http://pandaboard.org/content/resources/references>, 2011.
- [7] Programming language popularity. <http://langpop.com/>, 2011.
- [8] Teachingbox official website. <http://servicerobotik.hs-weingarten.de/en/teachingbox.php>, 2011.
- [9] Arduplane. <http://code.google.com/p/ardupilot-mega/>, 2012.
- [10] Gazebo. <http://gazebo-sim.org/>, 2012.
- [11] Official java website. <http://www.java.com/en/>, 2012.
- [12] Oracle website. <http://www.oracle.com/index.html>, 2012.
- [13] Procerus technologies - KESTREL autopilot. <http://www.procerusuav.com/productsKestrelAutopilot.php>, 2012.
- [14] PX4 autopilot. <https://pixhawk.ethz.ch/px4/>, 2012.
- [15] Wikipedia. <http://en.wikipedia.org>, 2012.
- [16] Python programming language - official website. <http://www.python.org/>, 2013.
- [17] sUAS news. <http://www.suasnews.com/>, 2013.
- [18] TIOBE programming community index. <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>, January 2013.
- [19] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, second edition edition, 2010.

- [20] Chris Anderson. Good primer on control theory and machine learning in UAVs. <http://www.diydrones.com/profiles/blogs/good-primer-on-control-theory-and-machine-learning-in-uavs>, November 2012.
- [21] J Andrew (Drew) Bagnell and Jeff Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings of the International Conference on Robotics and Automation*. IEEE, May 2001.
- [22] G. C. Balan, C. Cioffi-Reivilla, S. Luke, L. Panait, and S. Paus. MASON: A java multi-agent simulation library. In *Proceedings of the Agent 2003 Conference*, 2003.
- [23] Randal W. Beard. Quadrotor dynamics and control. Brigham Young University, February 2008.
- [24] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.
- [25] Haitham Bou-Ammar, Holger Voos, and Wolfgang Ertel. Controller design for quadrotor UAVs using reinforcement learning. In *Proceedings of the IEEE International Conference on Control Applications*. IEEE, 2010.
- [26] Geoffrey Bower and Alexander Naiman. Just keep flying: Machine learning for UAV thermal soaring. Stanford University Computer Science Department, 2006.
- [27] Cliff Click. Java vs C performance... again. <http://www.azulsystems.com/blog/cliff/2009-09-06-java-vs-c-performanceagain>, June 2009.
- [28] Drew Conway. Revisiting 'ranking the popularity of programming languages': Creating tiers. <http://www.drewconway.com/zia/?p=2892>, 2012.
- [29] John J. Craig. *Introduction to Robotics - Mechanics and Control*. Pearson Prentice Hall, third edition edition, 2005.
- [30] Mark Dupuis, Jonathan Gibbons, Maximillian Hobson-Dupond, Alex Knight, Michael Monfreda, and George Mungai. Design optimization of a quad-rotor capable of autonomous flight. Worcester Polytechnic Institute, April 2008.
- [31] J. M. Engel. Reinforcement learning applied to UAV helicopter control. Master's thesis, Delft University of Technology, July 2005.
- [32] Christian Felde. C++ vs java performance: It's a tie! <http://blog.cfelde.com/2010/06/c-vs-java-performance/>, June 2010.
- [33] Aaron Gamble, Peter Tan, Elijah Phillips, and Briget Dean. 2011 final year project - UAV test rig. University of Canterbury, November 2011.
- [34] Gabriel M. Hoffmann, Haomiao Huang, Steven L. Waslander, and Claire J. Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. American Institute of Aeronautics and Astronautics, 2007.
- [35] Louis Hugues and Nicolas Bredeche. Simbad: an autonomous robot simulation package for education and research. In *Proceedings of the Ninth International Conference on the Simulation of Adaptive Behaviour*, 2006.
- [36] Louis Hugues and Nicolas Bredeche. Simbad project home. <http://simbad.sourceforge.net/>, May 2011.



- [37] Eric N. Johnson and Michael A. Turbe. Modeling, control, and flight testing of a small ducted fan aircraft. *Journal of Guidance Control and Dynamics*, July 2006.
- [38] Stefan Krause. Java vs C benchmark. <http://www.stefankrause.net/wp/?p=4>, October 2007.
- [39] Sergei Lupashin, Angela Schoellig, Michael Sherback, and Raffaello D’Andrea. A simple learning strategy for high-speed quadrotor multi-flips. In *Proceedings of the IEEE Conference on Robotics and Automation*. IEEE, May 2010.
- [40] Jr Marcelo H. Ang and Vassilios D. Tourassis. Singularities of euler and roll-pitch-yaw representations. *IEEE Transactions on Aerospace and Electronic Systems*, May 1987.
- [41] James M. McMichael and Michael S. Francis.
- [42] Tom M. Mitchell. *Machine Learning*. The MIT Press and WCB/McGraw-Hill, 1997.
- [43] Andrew Y. Ng, H. Jin Kim, Michael I. Jordan, and Shankar Sastry. Inverted autonomous helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics*. MIT Press, 2004.
- [44] Tomaso Poggio and Federico Girosi. Networks for approximation and learning. In *Proceedings of the IEEE*, September 1990.
- [45] Angela Schoellig and R D’Andrea. Optimization-based iterative learning control for trajectory tracking. In *Proceedings of the European Control Conference*, August 2009.
- [46] Martin Sewell. Machine learning. Department of Computer Science, University College London, 2009.
- [47] Satinder P. Singh and Richard S. Sutton. Reinforcement learning with replacing eligibility traces. Kluwer Academic Publishers, 1996.
- [48] L. M. Sonneborn and F. S. Van Vleck. The bang-bang principle for linear control systems. *SIAM Journal on Control*, A 2.2, 1965.
- [49] Richard S. Sutton. Reinforcement learning FAQ. <http://http://webdocs.cs.ualberta.ca/~sutton/RL-FAQ.html>, April 2004.
- [50] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.
- [51] Brian Tanner and Adam White. RL-Glue: Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research*, September 2009.
- [52] John Valasek, James Doebbler, Monish D. Tandale, and Andrew J. Meade. Improved adaptive-reinforcement learning control for morphing unmanned air vehicles. In *IEEE Transactions on Systems, Man, and Cybernetics*. IEEE, August 2008.
- [53] Steven L. Waslander, Gabriel M. Hoffmann, Jung Soon Jang, and Claire J. Tomlin. Multi-agent X4-flyer testbed design: Integral sliding mode vs. reinforcement learning. In *Proceedings of the IEEE Conference on Intelligent Robots and Systems*, Edmonton, AB, Canada, August 2005. IEEE.